НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Система управління замовленням друку 3D-виробів на базі мікросервісів»

Виконав:
студент IV курсу, групи ІА-62
Панарін Валерій Ігорович _____

Керівник:
асистент кафедри АУТС
Дорога-Іванюк Олена Олександрівна _____

Рецензент:
доцент кафедри ПЗКС ФПМ, к.т.н
Цуркан Василь Васильович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__р.

ЗАВДАННЯ

на дипломний проєкт студенту

Панаріну Валерію Ігоровичу

1. Тема проєкту «Система управління замовленням друку 3D-виробів на базі мікросервісів», керівник проєкту Дорога-Іванюк Олена Олександрівна, асистент, затверджені наказом по університету від «7» травня 2020 р. №1081-с

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту: розроблена система повинна складатись щонайменше з чотирьох мікросервісів, повинна функціонувати при наявності від 2 гігабайт оперативної пам’яті на кожен з додатків, цільова платформа Linux/Windows Server.

4. Зміст пояснювальної записки: Опис предметної області. Огляд наявних рішень. Аналіз вимог до програмного забезпечення. Стек технологій та середовище розробки. Розробка системи. Тестування системи. Впровадження та використання системи.

5. Перелік графічного матеріалу: діаграма розгортання, схема зв’язку між вузлами, схема баз даних додатку, діаграма послідовностей процесу обробки замовлення.

6. Дата видачі завдання 30.04.2020.

Календарний план

№	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз предметної області	05.05.2020	
2	Аналіз наявних рішень	08.05.2020	
3	Аналіз вимог до програмного забезпечення	11.05.2020	
4	Вибір стеку технологій та середовища розробки	14.05.2020	
5	Розробка програмних додатків	20.05.2020	
6	Тестування системи	28.05.2020	
7	Оформлення проєкту	02.06.2020	
8	Подання готового проєкту	09.06.2020	

Студент Валерій ПАНАРІН

Керівник проєкту Олена ДОРОГА-ІВАНЮК

АНОТАЦІЯ

Панарін В.І. Система управління замовленням друку 3D-виробів на базі мікросервісів. КПІ ім. Ігоря Сікорського, Київ, 2020.

Проєкт містить 64 с. тексту, 9 рисунків, посилання на 15 літературних джерел, 4 конструкторських документи та один додаток.

Ключові слова: 3D друк, система замовлення 3D друку, мікросервіси.

Об'єктом дослідження в даному дипломному проєкті виступає система що дозволяє замовляти товари та послуги через інтернет.

Предметом дослідження виступає розробка системи управління замовленнями друку на 3D принтері з використанням мікросервісної архітектури.

Метою даної роботи є спрощення та надання зручного сервісу для замовлення друку на 3D принтері через інтернет для кінцевого споживача.

У результаті виконання даного дипломного проєкту було розроблено систему для управління замовленням друку 3D виробів на основі мікросервісів, що надає функціонал та засоби призначенні для її налаштування та використання.Під час аналізу наявних рішень у якості оптимального стека технологій для реалізації системи було обрано мову Typescript в поєднанні з фреймворком NestJs. Програмний продукт реалізовано у вигляді вебсервісу. Також в процесі підготовки до роботи та аналізу наявних рішень у сфері замовлення друку виробів на 3D принтері було досліджено стан цієї сфери та підтверджено актуальність розробки та необхідність подальшого розвитку отриманого програмного продукту.

ANNOTATION

Panarin V.I. Microservices-based management system for 3D print ordering. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2020

The project contains 64 pages of text, 9 figures, references to 15 literature sources, 4 design documents and 1 attachment.

Keywords: 3D printing, 3D print ordering system, microservices.

The object of diploma project is an automated system for ordering goods and service via the Internet.

The subject of this research is an automated system for 3D print ordering via Internet.

The purpose of this work is to simplify and provide a convenient service for ordering printing on a 3D printer via the Internet for the end user.

In the process of research of this diploma project, a microservice based system for managing the ordering of printing on a 3D printer was developed. System provides tools for self managing and configuring. When analyzing the available solutions, the Typescript language in combination with the NestJs framework was chosen as the optimal technology stack for the system implementation. The software product is implemented in the form of a web service.

Also in the process of preparation for work and analysis of existing solutions in the field of ordering printing products on a 3D printer was investigated the state of this area and confirmed the relevance of development and the need for further development of the software product.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА62.190БАК.005 ПЗ	Пояснювальна записка	64		
6	А3	ІА62.190БАК.005 Д1	Система управління	1		
7			замовленням друку			
8			3D-виробів на базі			
9			мікросервісів. Структурна			
10			схема. Зв'язок між вузлами			
11	А3	ІА62.190БАК.005 Д2	Система управління	1		
12			замовленням друку			
13			3D-виробів на базі			
14			мікросервісів. Діаграма			
15			розгортання додатку.			
16	А3	ІА62.190БАК.005 Д3	Система управління	1		
17			замовленням друку			
18			3D-виробів на базі			
19			мікросервісів. Схема баз			
20			даних додатку.			
21	А3	ІА62.190БАК.005 Д4	Система управління	1		
22			замовленням друку			
23			3D-виробів на базі			
24			мікросервісів. Діаграма			
25			послідовності для			
26			формування замовлення.			
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	**ІА62.190БАК.005 ТП**		
Розроб.		Панарін			Система управління замовленням друку 3D-виробів на базі мікросервісів. Відомість технічного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Дорога-Іванюк				Т	
Реценз.						КПІ ім. Ігоря Сікорського Каф.АУТС група ІА-62	
Н. Контр.							
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Система управління замовленням друку 3D-виробів на базі мікросервісів»

Київ – 2020 року

ЗМІСТ

ВСТУП

Зараз сфера замовлень послуг через мережу інтернет набирає все більшої популярності і кількість сервісів, що надають можливість замовити послугу через мережу інтернет за останні декілька років значно зросла. Такий спосіб замовлення послуг показав свою зручність та ефективність і допоміг багатьом компаніям збільшити об’єм продажу власних послуг та надати їм перевагу над конкурентами, що не надають аналогічного сервісу. Також в наш час значно зросла популярність друку виробів на 3D принтері.

Актуальність розроблення системи підтверджується появою великої кількості студій, що займаються друком виробів 3D принтері, в останні роки, а також велика кількість сервісів, що дозволяють замовити різноманітні послуги через інтернет.

Об'єктом дослідження в даному дипломному проєкті виступає автоматизована система, що дозволяє замовляти товари та послуги через інтернет.

Предметом дослідження виступає розробка системи управління замовленнями друку на 3D принтері з використанням мікросервісної архітектури.

Метою даної роботи є спрощення замовлення друку на 3D принтері через інтернет для кінцевого споживача. Окрім можливості замовити друк сервіс також являє собою онлайн платформою для публікації 3D моделей розроблених користувачами в CAD-системах. Розроблена система бере на себе управління формуванням замовлень та плануванням їх виробництва, надає інтерфейси для керування опціями для замовлень та інформації для користувачів, має систему модерації.

У процесі виконання даної роботи необхідно розв’язати наступні задачі:

— розробка структури додатку та способів виконання ним поставленої задачі;

— зробити систему розподіленою, а саме при розробці використати підхід з розбиттям додатку на мікросервіси;

закласти основу для подальшого розширення можливостей продукту та його розвитку;

Для розв’язання поставлених задач в роботі застосовуються такі методи:

— використання сучасного та правильно підібраного стека технологій;

— аналіз сфери 3D друку та суміжних з ним сфер з метою закласти в структуру проєкту можливість розширення системи для використання в них;

Додаток, отриманий як результат даної роботи має практичну цінність для підприємства, що займається друком 3D моделей та бажає збільшити кількість замовлень своїх послуг та автоматизувати процес обробки замовлень.

Дипломний проєкт складається з наступних розділів: вступ, опис предметної області, огляд наявних рішень, аналіз вимог до програмного забезпечення, стек технологій та середовище розробки, розробка системи, тестування системи, впровадження та використання системи, висновки, список використаних джерел, конструкторських матеріалів у вигляді діаграм. Графічна частина включає 4 кресленика формату А3. Загальний обсяг сторінок 64.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

В даному наведений короткий опис предметної області дипломного проєкту.

1.1 Вебсервіси

В наш час інтернет майже всі підприємства, починаючи з малих підприємств та закінчуючи великими корпораціями, використовують інтернет для реалізації своїх товарів, послуг тощо. Також створюється багато сервісів, які створюють нові ніші для бізнесу, такі як реклама в інтернеті, сервіси, що дозволяють користувачами опублікувати власний контент, такий як відео, музика, програмний код тощо та отримувати прибуток від продажу цього контенту чи від партнерських програм платформ, на яких вони публікуються. Створюється все більше платформ в мережі інтернет для замовлень різного роду послуг, таких як інтернет-магазини, вебсервіси для замовлення доставки їжі, сервіси продажу музики, стрімінгові сервіси, сервіси об’яв. Все це спрощує задачі бізнесу для реалізації та популяризації власних послуг завдяки тому що:

- в наш час майже всі користувачі мають інтернет та проводять в ньому досить велику частину свого вільного часу і бачать рекламу дуже часто.

- реклама в мережі інтернет є ефективною та дешевшою завдяки тому, що платформи, які надають послуги реклами в інтернеті, дозволяють робити рекламу персоналізовано і відповідно підвищують шанс того, що цільова аудиторія зацікавиться послугою, що рекламується.

- також в нас час є концепція вебресурсів на яких самі користувачі самі користувачі виробляють контент для платформи, там самі його споживають, такі як GitHub та аналогічні йому сервіси, різного роду фотобанки, дошки об’яв, такі як OLX(колишній Slando), інтернет аукціони, YouTube, Instagram, Facebook, TikTok тощо. Власники цих сервісів працюють в першу чергу над

тим, щоб надавати користувачам можливість свій контент публікувати, переглядати, слідкує за дотриманням правил сервісу (модерація), надає клієнтську підтримку.

Для таких платформ основним способом монетизації (отримання прибутку) є наступні способи:

- продаж реклами на платформі, так YouTube між переглядами відео робить рекламні вставки. Також сервіси вставляють рекламні банери на сторінках сервісу, чи беруть винагороду за те, щоб інші користувачі побачили їх контент у списку рекомендації (наприклад OLX).

- продаж платних облікових записів, які відкривають доступ до можливості не бачити рекламу при користуванню сервісом, чи знімають деякі обмеження, наприклад обмеження на кількість контенту, який може публікувати один користувач тощо.

- також існує концепція SaaS (Software as a Service), коли замість програмного продукту продається доступ до нього, наприклад як це робить Amazon Web Services.

1.2 3D друк

3D друк — це технологія виробництва виробів шляхом послідовного нарощування шарів матеріалу для отримання тривимірного об’єкта. 3D друк належить до категорії адитивних технологій та є одним з найпопулярніших, швидких та дешевих їх відгалужень. Для друку використовується пристрій, що має назву 3D-принтер. 3D принтери в наш час стають все більш популярними, як для фабричного виробництва, так і серед малих підприємств та приватних осіб, оскільки являються доступними та достатньо дешевими, пропонуючи привабливе співвідношення ціни та якості. Такий рівень популярності цієї технології спостерігається з 2003 року, оскільки технологія 3D друку стала достатньо дешевою та доступною. Технології 3D друку використовуються у

багатьох галузях виробництва, таких як:

— ювелірні вироби,

— автомобілебудування,

— аерокосмічна галузь,

— архітектура та будівництво,

— одяг та взуття

За основу для виробів на 3D принтері використовують моделі, розроблені у CAD-системах. Звичайно процес створення готових виробів потребує також додаткової обробки, як до так і після процесу друку виробу Це потрібно, щоб досягнути бажаного рівня якості кінцевого виробу.

Також на якість отриманого продукту може впливати програмне забезпечення, що використовується для керування процесом друку на принтері.

Серед основних способів створення виробів на 3D принтерах є:

— SheetLamination — це пошарове склеювання або з’єднання іншими способом листових матеріалів;

— BinderJetting — це пошарове розбризкування або нанесення струменем матеріалу, що зв'язує.

— MaterialJetting — це пошарове розбризкування конструкційного матеріалу.

— PowderBedFusion — це підхід, при якому відбувається розплавлення матеріалів, з яких було попередньо сформовано шар, після чого відбувається спікання деяких його частин.

— VatPhotopolymerization — це підхід, при якому пошарово відбувається фотополімеризація матеріалу у ванні або затвердіння фотополімерних смол. Цей підхід часто використовується для виробництва деталей, що мають малі розміри (менш як 1 см) та надаючи при цьому достатньо високий рівень деталізації.

— MaterialExtrusion — найпопулярніша наразі технологія, суть якої полягає в пошаровому нанесені матеріалу через екструдер. Зазвичай матеріал

для цього типу друку являє собою пластик у формі нитки товщиною 200-500 мікрон.

В даному проєкті розроблена система орієнтована на використання для замовлень друку на 3D принтері, але також може використовуватись і для обробки замовлень пов’язаних з ЧПК станками. Оскільки 3D друк це лише одна технологія з сімейства адитивних технологій, то потенційно при подальшому розвитку системи її можливо модифікувати для роботи з замовленнями з інших сімейств адитивних технологій.

1.3 Висновки розділу

З огляду інформацію приведену в даному розділі можна зробити такі висновки, що 3D друк є лише однією з технологій в сімействі адитивних технологій і потенційно подальший розвиток системи, що розробляється, можна спрямувати в сторону додавання до асортименту послуг роботу з іншими технологіями з сімейства адитивних технологій.

Також можна побачити, що існує багато способів друку 3D виробів та відповідно різних видів принтерів. Даний факт потрібно враховувати при подальшій розробці продукту.

Також було проаналізовано ситуацію, актуальну даний момент, відносно наявних вебсервісів на ринку, їх особливостей та способів отримання прибутку від їх впровадження.

2 ОГЛЯД НАЯВНИХ РІШЕНЬ

На сьогодні друк 3D виробів став дуже популярним у якості хобі або як засіб для створення деталей для виробів, які потрібні у дуже обмеженій кількості. Створюється багато студій, що займається друком, а також 3D принтер зараз можна купити самому. Але у всього цього є деякі недоліки. По-перше, далеко не кожен може дозволити собі принтер, який може задовольнити його вимоги, також дуже часто користувачу потрібно видрукувати якусь деталь лише в єдиному екземплярі і робить він це дуже не часто і тому він не бачить потреби в тому, щоб придбати для цього принтер. По-друге, 3D принтер може бути досить складним в обслуговуванні. Тому для багатьох користувачів простішим рішенням є просто замовлення друку у студії чи людини яка має 3D принтер. При замовленні 3D у спеціалізованої студії також є недолік у тому, що іноді люди бояться і вирішують, що друк якоїсь малої деталі у єдиному екземплярі просто не вартує того, щоб зв'язатися зі студією чи шукати на спеціалізованих форумах людей, що готові видрукувати та вислати деталь, чи передати її особисто. Оскільки проєкт даної дипломного проєкту націлено на автоматизацію замовлення друку через інтернет, то основними аналогами виступають сайти студій, що займаються 3D друком.

2.1 Сайт для замовлення 3D друку та обладнання 3dreams.com.ua

3dreams.com.ua — це сайт фірми, що займається 3D друком. При переході на їх сайт можна побачити, що фірма займається не тільки друком, а й продажем обладнання для друку.

Сама фірма описує процес реалізації послуги наступним чином: розрахунок вартості, передоплата, фото готового продукту та оплата, відправлення. Користувач повинен брати участь у кожному з етапів, що може досить сильно розтягувати часові рамки виконання замовлення.

Для замовлення користувачу потрібно заповнити форму, вказавши особисті дані та завантаживши файли виробів, які він бажає видрукувати, після чого дочекатись доки з ним зв'яжеться оператор. На окремій сторінці користувач може розрахувати приблизну вартість друку моделі, завантаживши її.

Також сервіс має каталог обладнання для друку 3D виробів. Для відправлень сервіс користується послугами компанії Нова Пошта. У формі замовлення адреса не вказується, адреса вказується лише після завершення друку товару. Перед відправленням користувачеві надсилається фото для того, щоб він підтвердив відправлення та оплатив товар.

2.2 Сайт для замовлення 3D друку та обладнання 3ddevice.com.ua

3ddevice.com.ua — це ще один сервіс для замовлення друку на 3D принтері. При переході на їх сайт одразу показуються розцінки та пропонується заповнити форму для того, щоб оператор зміг зв’язатись з ним та сформувати замовлення.

Як і попередній аналог сервіс пропонує каталог обладнання для 3D друку такого як 3D принтери, 3D сканери, 3D ручки та матеріали для друку.

Окрім друку та продажу обладнання сервіс пропонує додаткові послуги такі як:

— 3D сканування,

— лиття пластмас у силікон,

— 3D моделювання,

— дрібносерійне виробництво,

— макетування,

— проєктування.

2.3 Недоліки наявних рішенньь

Основними недоліками аналогів є:

— складний процес замовлення, що потребує зв’язку з виконавцем;

— користувач повинен завантажувати лише свою модель для друку, що не дозволяє звичайним користувачам надавати свої моделі у якості моделей доступних для друку будь-кому.

2.4 Висновки розділу

На даний момент на ринку не представлено рішень для замовлення 3D друку, яке змогло би зробити замовлення максимально швидким та простим та мінімізувати потребу у прямих перемовинах між замовником та реалізатором.

Ідея даного проєкту полягає у тому, щоб максимально спростити та автоматизувати процес замовлення там отримання замовлення замовником, але не обмежується лиши цим. Даний проєкт буде являти собою платформу, яка крім простого продажу послуги друку буде реалізувати ще й ідею аналогічну фотобанкам. На цій платформі користувачі можуть публікувати 3D моделі, які вони самі розробили, в каталозі, який буде доступним на сторінці сервісу і кожду модель інші користувачі зможуть замовити для друку, вказавши потрібні їм, розміри, матеріали тощо. В той самий час користувачі можуть не публікувати свої моделі, а просто замовити їх друк, вказавши потрібні параметри, після чого сервіс обробить їх запит і почне друк.

Як можна помітити в даному проєкті вебсервіс, що буде розроблено розрахований не на лише на публікацію моделей, але й на замовлення їх друку. Тобто сервіс також сам буде займатись друком, тому система буде мати панель адміністратора, де оператори будуть займатись обробкою замовлень, зв’язком з клієнтами для уточнення деталей, надавати сервіс підтримки, актуалізувати списки доступних матеріалів, займатися модерацією опублікованих моделей.

3 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основними вимогами до програмного забезпечення можна розділити на функціональні на нефункціональні.

Функціональні вимоги відповідають за можливості, що надає системи при роботі з нею, в той час як нефункціональні вимоги відповідають за деякі особливості реалізації системи, а також її якісні показники.

Даний етап є дуже важливим, оскільки правильно сформовані функціональні та нефункціональні вимоги напряму впливають на процес розробки, прийняття рішень під час розробки та якість отриманого програмного продукту.

Нижче описано та проаналізовано обидва види вимог.

3.1 Функціональні вимоги

Основним призначенням системи є спрощення процесу замовлення послуги та її обробки. При детальнішому розгляді вимог можна сформувати наступні вимоги:

— система повинна дозволяти користувачам завантажувати моделі, розроблені в CAD-системах;

— користувачі повинні мати можливість переглядати інформацію про наявні в каталозі 3D моделі, а також публікувати власні;

— кожна модель повинна проходити модерацію операторами системи на предмет коректності завантаженої моделі як в технічному плані так і в плані дотримання правил заданих самим сервісом;

— користувач повинен мати можливість сформувати кошик з набором моделей та обрати потрібні йому опції друку, які система зможе реалізувати;

— система повинна надавати інструменти для обробки замовлень: отримання даних про замовлення, зміни його статусу в процесі виконання.

3.2 Нефункціональні вимоги

З не функціональних вимог можна виділити наступне:

— розроблена система повинна легко масштабуватись при подальшому її розвитку;

— система повинна мати спосіб захисту своїх даних

— система повинна мати розподіл функціональності по ролях серед її користувачів.

— система повинна мати достатню швидкодію та можливості для розширення, щоб встигати обробляти потік замовлень, що будуть їй поступати.

3.3 Висновки розділу

При аналізі наведених вище вимог можна сказати, що система повинна надавати зручний графічний інтерфейс, який можна легко розширювати при потребі та який надає можливості для роботи з системою лише через нього як для оператора система, так і для кінцевого користувача. Мати способи для автентифікації та авторизації для всіх мікросервісів з використанням лише одного токену доступу, вміти реагувати по-різному в залежності від того, хто з нею працює, тобто мати систему розподілу прав між користувачами та мати розділення на кінцевих користувачів системи та її операторів.

4 СТЕК ТЕХНОЛОГІЙ ТА СЕРЕДОВИЩЕ РОЗРОБКИ

Нижче описано стек технологій та програмного забезпечення для розробки додатку.

4.1 Фреймворк для розробки браузерних додатків

Додаток являє собою вебдодаток і для роботи з сервісом користувач буде використовувати графічний клієнт, який працює в браузері. Самих графічних клієнтів є два: для користувачів та операторів.

Сучасні графічні клієнти повинні реалізувати багато функціоналу, який досить важко реалізувати з таким підходом, як це було колись, тобто коли сервер повертає користувачу html сторінку, на якій зібрано певний функціонал, який стилізується за допомогою CSS та JavaScript. Недоліком таких додатків є те, що вони напряму пов’язані з вебсервером, який також зазвичай зберігає бізнес-логіку і формує динамічні сторінки. В наш час широко використовується SPA (Single page application) підхід для створення графічних клієнтів. Особливістю його є те, що клієнт є окремим додатком, який має власний внутрішній стан, може працювати з одразу декількома вебсерверами. Він розгортається окремо від основного сервера, а також бере на себе відповідальність за візуалізацію контенту у браузері, навігацію по сайту тощо. Такий підхід дозволяють реалізувати такі frontend-фреймворки, як AngularJs, VueJs. При розробці таких додатків зазвичай більшу частину додатку складає JavaScript, саме він зберігає в собі логіку візуалізації контенту та спілкування з сервером. Для розробки вебклієнтів обрано VueJs версії 2.6.10, який є повноцінним фреймворком для створення SPA додатків. Даний фреймворк за основу підхід MVVM (Model, View, Viewmodel), але реалізує його не в повному обсязі.

Також при розробці цих додатків широко використовується архітектура flux, розроблена компанією Facebook. Цей підхід призначений для спрощення

роботи з потоком даних у додатку та базується на ідеї однонапрямного потоку даних. Для реалізації такої архітектури існує бібліотека vuex, яка розрахована на роботу з додатками, що розробляються з допомогою VueJs.

Для того, щоб браузерний додаток поводився так, як і звичайний сайт, тобто мав різні сторінки, по якім можна переходити за посиланнями, використана бібліотека vue-router. Її призначення полягає у тому, що вона синхронізує окремі екрани з адресою, на яку посилається користувач, а також працює з історією браузера, що дозволяє переходити по історії перегляду сторінок у двох напрямках, також виступає як посередник для роботи с query параметрами адреси сторінки.

4.2 Фреймворк для розробки мікросервісів

При розробці мікросервісів використовується фреймворк NestJs та мова Typescript. NestJs - це фреймворк для розробки серверних додатків для платформи NodeJs, що широкий вибір інструментів. Основною його перевагою є те, що він надає готову архітектуру для розробки додатку, що пришвидшує розробку, та допомагає уніфікувати підхід до реалізації різних компонентів. Він комбінує в собі об’єктноорієнтований, функціональний та функціональний реактивний підходи.

Даний фреймворк не є аналогом таких розповсюджених рішень як ExpressJs, KoaJs, Fastify, а являє собою набір компонентів, що працюють з використанням цих серверних фреймворків. По замовчуванню доступні ExpressJs та Fastify, але при потребі фреймворк дозволяє створити адаптер для будь-якого аналогічного серверного фреймворку. Сам NestJs допомагає структурувати код, зробити його модульним, а також адаптує багато розповсюджених рішень під свою екосистему.

Мікросервіси мають модульну архітектуру та використовують інверсію залежностей, але на відміну від графічних клієнтів, іос контейнер постачається

фреймворком додатку.

4.3 Мова розробки

Вище зазначені фреймворки допомагають організувати код додатку, а також беруть на себе частину роботи з візуалізацією контенту, роботі з історією так реалізацією деяких архітектур та шаблонів. При розробці додатків, які мають багато складної логіки іноді є не дуже зручним використання мови JavaScript оскільки вона має багато недоліків, основними є:

— відсутність строгої типізації. Строга типізація набагато полегшує розробку програми, дозволяючи більш чітко задавати структуру її частин;

— JavaScript дозволяє реалізувати багато парадигм програмування, але майже всі парадигми вона дозволяє реалізувати або не в повному об’ємі, або не зручним для програміста способом;

Тому для розробки коду додатків, як і клієнтів, так і мікросервісів використана мова TypeScript, розроблена компанією Microsoft. Особливістю цієї мови є то, що вона розширює синтаксис JavaScript, а також має власний компілятор, який компілює програму написану на мові Typescript в javascript код. Наразі ця мова стає все більш популярною та надає багато переваг. В першу чергу мова допомагає більш зручно використовувати об'єктноорієнтований підхід, додаючи:

— модифікатори доступу public, protected, private. Мова JavaScript не має такого механізму;

— абстрактні класи та методи. JavaScript не має механізму, що дозволяє засобами мови показати, що клас є абстрактним і які саме частини потрібно реалізувати;

— інтерфейси. Оскільки в JavaScript немає строгої типізації, то не можна описувати структури, які передаються у параметрах методів, функції при об’явленні змінних чи констант;

— полегшує роботу з поліморфізмом. JavaScript дозволяє реалізувати поліморфізм, але він не є очевидним. Typescript є розширенням JavaScript, а не окремою мовою, тому вона не може повноцінно ввести даний механізм, але дозволяє описувати сигнатури поліморфних методів та функцій, щоб при зверненні до них можна було побачити їх перевантаження та знати тип результату, що буде повернено при їх викликах;

— рушій JavaScript V8, розроблений компанією Google, що широко використовується браузерами (наприклад Google Chrome, Opera, Comod Dragon, Vivaldi) та платформою NodeJs, більш ефективно працює зі статично типізованим кодом (тип даних у змінній не змінюється під час виконання програми) і Typescript допомагає дотримуватися цього підходу, тим самим допомагає підвищити швидкість роботи програми;

4.4 Збірка проєкту

Оскільки сучасні браузери не мають підтримки TypeScript, то для збірки додатків буду використано Webpack. Даний продукт працює на платформі NodeJs та вирішує широкий спектр задач, що виникають при розробці вебдодатків. Основними його функціями, що використано при розробці:

— компіляція Typescript в JavaScript;

— бандлінг, тобто склейка багатьох файлів в один. Це дозволяє при завантаженні клієнтського javascript коду завантажувати його як один файл, а отже зменшити кількість запитів на сервер, що підвищує швидкість завантаження сторінки;

— мініфікація. Оскільки HTML, CSS, та Javascript не мають жорстких правил табуляції та перенесення рядка, то зайві перенесення можна видаляти, також багато змінних, що не є глобальними можна замінити на коротші, не змінюючи при цьому логіку роботи програми, що також зменшує розмір вихідного файлу;

— компіляція Sass (в цьому випадку варіант з синтаксисом scss) в CSS;

— перетворення javascript коду в старіші стандарти специфікації ECMA Script для підтримки старими браузерами.

4.5 Система управління базами даних

На роль системи управління базами даних (СУБД) для всіх мікросервісів було обрано СУБД з відкритим вихідним кодом Postgresql версії 12.0.

Причиною вибору даної СУБД слугують наступні фактори:

— безкоштовність та відкритість вихідного коду;

— дотримання стандарту SQL: актуальна на наданий момент версія 12 дотримується що найменше 160 з 179 обов'язкових вимог для відповідності SQL:2016 Core;

— додаткові розширення синтаксису SQL та типів даних;

— система може виконувати одночасно більше однієї операції запису.

Як аналоги були також розглянуті такі СУБД, як SQLite, MySql, Oracle та Sql Server.

Окрім реляційних баз даних також були розглянуті документовані бази даних, такі, як MongoDb, але при більш детальному дослідженні специфіки роботи з ними, були встановлено, що бази такого краще підходять для зберігання різнорідних даних, та можуть проявляти себе гірше, ніж реляційні бази, якщо структура бази жорстко задана.

4.6 Протоколи та технології для спілкування компонентів системи

Для комунікації між компонентами додатку є два основних способи: запит-відповідь (рисунок 4.1) та комунікація, що базується на публікаціях та підписниках.

Стиль запит-відповідь корисний, коли виникає потреба в обміні

повідомленнями між різними зовнішніми службами. За допомогою цієї парадигми можна бути впевненими в тому, що служба фактично отримала повідомлення та обробила його. Такий підхід активно використовується як для спілкування мікросервісів з мікросервісами, так і клієнтів в мікросервісами. В першу чергу це дозволяє зробити запит на дані та отримати їх. Для такого стилю спілкування в даному проєкті використовуються протоколи TCP та HTTP зокрема.

Для роботи підходу запит-відповідь в мікросервісах створюється два канали: перший відповідає за передачу даних від сервісу, а другий очікує на вхідні повідомлення.

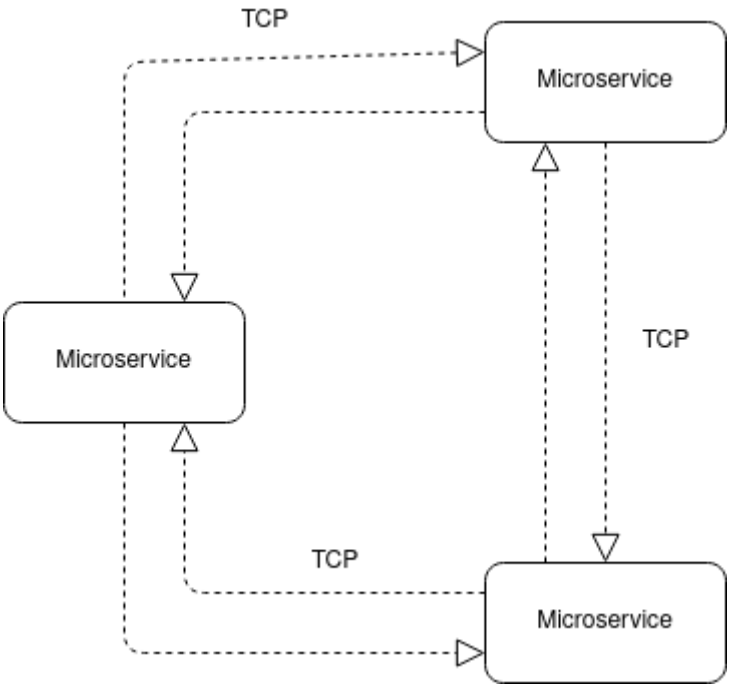

Рисунок 4.1 — Комунікація в стилі запит-відповідь

Спілкування за допомогою повідомлень та підписників актуальне тоді, коли двостороння передача даних між сервісами не потрібна. Такий підхід дозволяє зменшити навантаження на сервіси оскільки не потребує підтримання двох каналів для передачі даних.

Серед систем, що базуються на публікації подій основними рішеннями є

Redis, Message Queuing Telemetry Transport(MQTT), NATS, RabbitMQ та gRPC.

Redis - це програмне забезпечення з відкритим вихідним кодом, що являє собою сховище для зберігання даних в пам’яті, може використовуватись як база даних, кеш чи брокер повідомлень. Дане програмне забезпечення реалізує парадигму публікація/підписка (рисунок 4.2). Опубліковані повідомлення категоризуються по каналах, а той хто опублікував повідомлення не знає, чи прочитав його хтось з підписників і чи є такі взагалі.

Redis не гарантує того, що повідомлення буде прочитане кимось з підписників, оскільки працює за стратегією fire-and-forget, тобто після публікації повідомлення і передачі його всім підписникам. що прослухують повідомлення в даний момент, повідомлення видаляються.

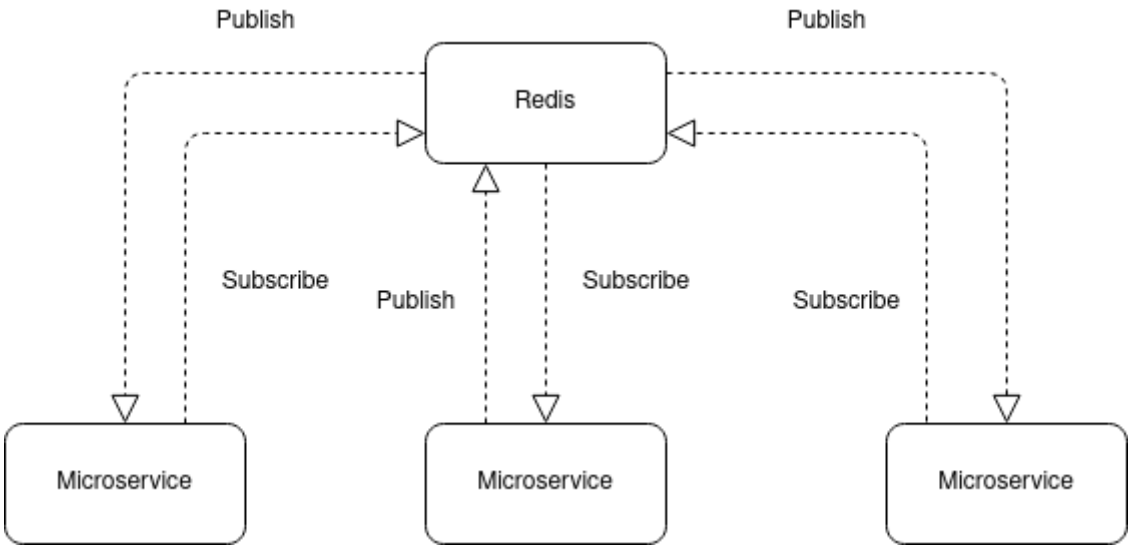

Рисунок 4.2 — Комунікація в стилі повідомлення/підписка

MQTT - це протокол підключення "машина до машини" (M2M) / "Інтернет речей". Він розроблений у якості надзвичайно легкого відкритого протоколу для спілкування в парадигмі публікації / підписки на повідомлення. В першу чергу даний протокол орієнтований з ненадійним з’єднанням або низькою пропускною здатністю. Наприклад, він використовувався в сенсорах, що спілкуються з брокером по супутниковому каналу зв'язку, при періодичних

комутаційних зв’язках з постачальниками медичних послуг, а також у ряді домашньої автоматизації та невеликих сценаріїв пристроїв. Він також ідеально підходить для мобільних додатків, оскільки він має низьке енергоспоживання, мінімізовані пакети даних та ефективне розповсюдження інформації на один або кілька приймачів.

NATS - це проста, безпечна та високоефективна система обміну повідомленнями для хмарних додатків, IoT-повідомлень та мікросервісів. NATS є програмним забезпеченням з відкритим вихідним кодом. Сервер NATS написаний мовою програмування Go, але бібліотеки клієнтів для взаємодії з сервером доступні для великої кількості основних мов програмування, включаючи JavaScript. NATS підтримує доставку якнайменше одного разу та як мінімум одного разу. Він може працювати де завгодно, від великих серверів та додатків, що розміщені в хмарах та навіть пристрої інтернету речей. NATS реалізує парадигму комунікації як запит-відповідь, так і підписка/публікація.

Також однією з корисних особливостей NATS є можливість використовувати синтаксис, що схожий на регулярні вирази для підписки на події.

RabbitMQ - це брокер повідомлень з відкритим кодом, який підтримує безліч протоколів обміну повідомленнями. Він може бути розгорнутий у розподілених інфраструктурах, щоб задовольнити великі вимоги до можливостей масштабування та високої доступності. Крім того, це найпоширеніший брокер повідомлень, який використовується у всьому світі для невеликих стартапів та великих підприємств.

gRPC - це сучасний, з відкритим кодом, високоефективний фреймворк для remote procedure call (RPC), який може працювати в будь-якому середовищі. gRPC ефективно налагоджувати підключення служб між центрами обробки даних за можливістю балансування навантаження, відстеження та перевірки стану коректності роботи встановлених з’єднань та аутентифікації.

Як і в багатьох системах RPC, gRPC заснований на концепції визначення

функцій (методів) в мікросервісах, які можна викликати віддалено. Для кожного методу визначається визначаєте параметри, що передаються, та типи даних що повертаються. Послуги, параметри та типи повернення визначаються у файлах .proto за допомогою механізму буферів протоколів нейтральних до мови із відкритим вихідним кодом від Google.

4.6 Контейнеризація компонентів додатку

Оскільки додаток є розподіленим, а його мікросервіси не пов’язані між собою та можуть мати різні вимоги до платформи, на якій працюють, то при розробці додатку використано контейнеризацію. Контейнеризація — це метод віртуалізації, при якому ядро операційної системи підтримує кілька ізольованих просторів користувача. Ці екземпляри просторів для користувача виглядають як окремі операційні системи. Це дозволяє ізолювати всі додатки інфраструктури додатку такі, як окремі мікросервіси, бази даних, сервери для зберігання статичних файлів в окремих контейнерах. Контейнеризація відрізняється від апаратної віртуалізації тим, що апаратна віртуалізація емулює також апаратне забезпечення, тому на ній можна запускати різні системи. Контейнеризація передбачає роботу операційних систем в цьому контейнері на тому ж ядрі, що і хост. Такий підхід став широко використовуватись завдяки тому, що вона мінімізує витрати на віртуалізацію та дозволяє працювати окремим контейнерам з мінімальними затримками, виключає проблеми з розв'язання проблем з залежностями, які виникають при розгортанні великої кількості додатків на одній операційній системі та дозволяє почати розробку без витрат часу на налаштування оточення додатку та його переналаштування при змінах в інфраструктурі. Також багато сервісів AWS дозволяють спростити розгортання додатків в контейнерах.

Системи для CI/CD такі, як, наприклад, Gitlab CI також широко використовують контейнеризацію для автоматизованої збірки, тестування та

розгортання програмних додатків. Для даного проєкту програмним забезпеченням для контейнеризації було обрано Docker. Для локальної розробки додатку все програмні компоненти працюють всередині контейнерів, які керуються за допомогою програмного забезпечення docker-compose, який дозволяє автоматизувати старт багатьох контейнерів з додатками, налаштувати їх зв’язок з хост системою та об’єднати їх у внутрішню мережу для спілкування між собою. Таким чином мікросервіси ізольовано в контейнери, які у внутрішній мережі спілкуються між собою та базами даних, а також відкривають потрібні порти хост системі для отримання запитів від неї.

Контейнери Docker будуються з допомогою спеціальних конфігураційних файлів, які можуть бути написаними самими користувачами. Таким чином надається можливість додаткового налаштування програмного забезпечення в контейнері. Наприклад, якщо для роботи скриптів, написаних на мові python потрібні додаткові бібліотеки чи програмні пакети, то створивши власний контейнер, що базується на контейнері в якому вже є встановлений python, можна отримати повністю налаштований по замовчуванню контейнер, який надалі буде готовий до роботи одразу після запуску.

Більшість контейнерів Docker працюють на легких операційних системах сімейства Linux з мінімальною кількістю залежностей для пришвидшення їх роботи. В основному для цих цілей використовується Alpine Linux та Ubuntu Linux.

Для зберігання та обміну готовими налаштованими контейнерами існує інтернет платформа DockerHub, яка дозволяє завантажувати готові контейнери з інтернету, а також публікувати власні. Таким чином при розробці в команді для розгортання всієї інфраструктури додатку на комп’ютері розробника достатньо мати встановлені в системі docker та docker-compose, після чого при виконанні команди «docker-compose up» всі потрібні залежності будуть завантажені з віддаленого сервера та запущені на локальній машині, одразу після чого можна починати розробку.

4.7 Керування залежностями

Оскільки додатки, що розробляються потребують для своєї як достатньо велику кількість бібліотек, так і програмного забезпечення, потрібного для їх компіляцій та запуску було використано пакетний менеджер для платформи NodeJs для керування залежностями.

Наразі наявні два основних пакетних менеджери: npm та yarn.

Npm — це старий, перевірений часом, пакетний менеджер для NodeJs, що постачається одразу з платформою та є її пакетним менеджером по замовчуванню.

Yarn — це більш нова альтернатив npm, розроблена компанією Facebook. Основною перевагою даного пакетного менеджера є менша кількість помилок при встановленні нових залежностей та оновленні версій пакетів до більш нових. Саме цей пакетний менеджер було обрано для локальної розробки.

Залежності проєкту поділяються на ті, що потрібні в процесі розробки, та ті, що потрібні в процесі використання. До першої категорії відносяться препроцесори, збірники проєктів webpack та rollup, монітори файлів такі, як nodemon тощо. До другої категорії належать залежності такі як vue, vuex, router та інші бібліотеки та фреймворки, що використовуються в вихідному коді проєкту.

Існує два способи встановлення залежностей: глобальний та локальний. Для розробки рекомендується використовувати локальний для проєкту спосіб збереження залежностей.

4.8 CI/CD та контроль версій

При розробці було використано систему контролю версій git. Ця система наразі є найпопулярнішою та майже витіснила своїх основних конкурентів Mercurial та Subversion. Дана система контролю версій обрана тому, що наразі існує багато ресурсів, що працюють саме з нею та надають послуги віддаленого зберігання репозиторіїв та CI/CD. При розгортанні проєкту в майбутньому та його подальшої розробки та підтримка дуже важливими відходами в наш час, оскільки автоматизують розгортування, публікацію, компіляцію та тестування програмного забезпечення, при цьому інтегруючись з системою контролю версій.

Принцип роботи CI/CD можна розібрати на прикладі Gitlab-Ci, що інтегрована з GitLab - відкритою платформою для віддаленого зберігання вихідного коду, що керується системою контролю версій git. GitLab-Ci постачає GitLab Runner, що використовує контейнери докер та конфігурації описані в спеціально виділених файлах проєкту для запуску сценаріїв розгортування, публікації, компіляції та тестування. Користувачі сами описують умови, при яких запускається той чи інший сценарій. Основними тригерами для цього виступають зміни в певних файлах чи директоріях проєкту, чи окремих гілках системи контролю версій, розгортування, публікацію, компіляцію та тестування програмного забезпечення.

4.9 Шаблон управління станом Vuex

При розробці графічних клієнтів було використано шаблон управління станом vuex, що розроблений авторами фреймворку Vuejs для використання разом зі своєю бібліотекою.

Vuex - це підхід, що замінює собою Model-View-Controller (MVC). Причиною вибору саме цього підходу є те, що MVC гірше проявляє себе при

розробці додатків, що мають велику кодову базу, чи таку, яка швидко росте. Даний підхід є аналогом flux від компанії Facebook направлений на те, щоб зробити потік даних в додатку однонапрямним, але має деякі відмінності від flux.

Основними компонентами цього шаблону є:

— геттер,

— сховище,

— мутація,

— дія

Зв’язок перерахованих вище компонентів показано рисунку 4.3.

Особливістю такої архітектури є однонаправленість потоку даних. Це досягається завдяки тому, щоб замість прямих викликів і отримання результатів, створюється система з компонентів для управління внутрішнім станом, яка надає інтерфейс, в даному випадку шину подій, для зміни внутрішнього стану.

Це потрібно тому, що при наявності компонентів в інтерфейсі, які базуються на спільному стані виникає проблема з синхронізацією їх стану, оскільки:

— Декілька представлень можуть залежати від однієї й тієї ж частини стану програми.

— Дії з різних представлень можуть впливати на одні й ті ж частини стану програми.

Першу проблему можна вирішити шляхом передачі одні і тих же параметрів з кореневого елемента у внутрішні, але це ускладнює код програми, та будь-які зміни в ньому.

Другу проблему можна вирішити шляхом прямих викликів між компонентами представлення, але це також створює велику складність при розробці, особливо, якщо потрібно синхронізувати стан сусідніх представлень.

Тому для розв'язання цієї проблеми загальний стан винесено в сховище,

що саме реалізує шаблон «Одинак» (Singleton), тобто є глобальним і завжди існує лише в одному екземплярі. Сховище — це контейнер для зберігання внутрішнього стану додатку та бізнес-логіки.

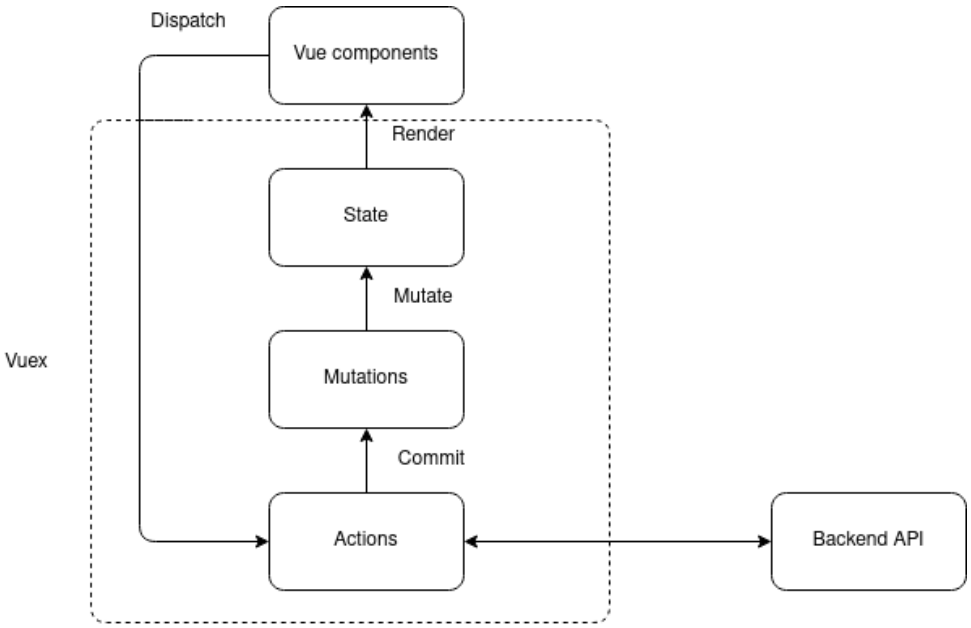

Рисунок 4.3 — Взаємодія компонентів

Для спрощення роботи з компонентами, які потребують додаткової обробки чи обрахувань для представлення використовують гетери. Гетери — це допоміжні функції, задача яких виймати дані зі сховища, обробляти їх та повертати представленню.

Мутації — це функції, роль яких полягає в зберіганні даних в сховище. Використання мутацій спрощую налагодження роботи програми, оскільки дій зазвичай є асинхронними і в них проблематично відстежувати роботу дій зі сховищем.

Дія — дуже схожа на мутацію за виключенням того, що:

— Замість прямої взаємодії зі сховищем дія використовує мутації;

— Дія може бути асинхронною.

Дії являються інтерфейсом для представлень, які представляння викликають через шину подій. В діях можна виконувати виклики на віддалений

сервер, чи мутації для переходу між різними станами. Мутації можливо використовувати у якості синхронних дій, але це не рекомендовано.

Оскільки при розробці додатків використовується об’єктноорієнтований підхід, то з класичного вигляду, запропонованого авторами шаблону, модулі vuex були перетворені на класи, за допомогою сторонньої бібліотеки, але при цьому зберегли концепцію підходу: дії замінені на публічні методи. які ніколи не повертають результат, мутації — приватні методи для зміни внутрішнього стану, геттери замінені на геттери класів в Typescript, а сховище замінено приватними полями класу. Сам модуль, у вигляді класу, являє собою Singleton.

4.10 Комунікація між програмними компонентами

Для комунікації між клієнтом та сервером використано підхід, що називається REST (Representational State Transfer). Це архітектурний стиль, що широко використовуються для взаємодії компонентів додатку розподіленого в мережі. Вебслужби, що дотримуються цього принципу називаються RESTful. Основними вимогами до RESTful служб є:

— використання моделі клієнт-сервер;

— відсутність стану;

— кешування. Це зменшує потребу в запитах на сервер. Але при цьому сервер повинен позначати, які саме відповіді можна кешувати;

— маніпуляція через представлення: дані з сервера передаються не у тому ж вигляді, що зберігаються на сервері;

— повідомлення повинні самі описувати, як їх потрібно обробляти;

— клієнт не знає, що доступна певна операція над ресурсом, якщо попередньо не отримав інформацію про це з попередніх викликів;

Такий підхід допомагає підвищити швидкість роботи додатку, оскільки замість запиту html сторінки від сервера клієнт робить запит лише на дані з сервера, чи запити на їх зміну, передаючи лише потрібний мінімум інформації,

також зробити спосіб взаємодії більш уніфікованим.

Взаємодію компонентів системи наведено в ІА62.190БАК.005 Д1. На даній діаграмі описана взаємодія між браузером, що завантажує код графічного клієнта зі статичного сервера та його спілкування з мікросервісам, а також мікросервіса з базою даних та файловою системою. У якості мікросервіса на діаграмі зображено файловий мікросервіс. Взаємодія з іншими мікросервісами аналогічна до наведеної на діаграмі.

Цей підхід використано не тільки при спілкуванні графічного клієнта та сервера, а також і для спілкування мікросервісів між собою. Графічний клієнт спілкується з мікросервісами, використовуючи протокол HTTP, дані між ними передаються у форматі JSON (окрім медіафайлів).

При спілкуванні мікросервісів використовується протокол TCP, а данні також передаються у форматі JSON. При спілкуванні мікросервісів з клієнтами, клієнти повинні проходити аутентифікацію, передаючи в HTTP запитах на сервер токен, що згенеровано за стандартом JWT (JSON Web Token). Це відкритий стандарт для генерації токенів для доступу. За генерацію та перевірку токенів відповідає окремий мікросервіс для аутентифікації, що їх генерує, та перевіряє. При цьому мікросервіси самі реалізують механізм авторизації (перевірка прав на виконання певних дій). Діаграму розгортування, яка також дозволяє побачити як мікросервіси спілкуються між собою, зокрема, взаємодію з сервісом автентифікації наведено в ІА62.190БАК.005 Д2.

4.11 Інверсія управління

При розробці також використовується інверсія управління. Цей принцип є важливою частиною при використанні об'єктноорієнтованого підходу. Він дозволяє зменшити зачеплення у програмі. Зачепленням називається ступінь зв'язаності компонентів програми. При розробці додатків слід уникати сильного зачеплення, оскільки це впливає на ступінь розуміння логіки окремих

компонентів, а також ускладнює тестування коду автоматизованими тестами. Для розв'язання цієї проблеми існують різні інструменти та підходи. Для даного проєкту була обрана інверсія залежностей. Суть цього підходу полягає в підміні залежностей компонентів від конкретної реалізації залежністю від інтерфейсів та використання системи, що буде займатись наданням окремих залежностей тоді, коли це потрібно. Такий підхід використано як при розробці коду мікросервісів, так і графічних клієнтів. У випадку з графічними клієнтами використовується бібліотека Inversify, що бере на себе роботу з залежностями. Але при розробці разом з VueJS виникає проблема з постачанням залежностей в окремі компоненти, що напряму пов’язані з візуалізацією контенту, оскільки їх не можна помістити в Ioc контейнер, бо створенням їх екземплярів займається сам фреймворк і він не постачає механізмів для передачі залежностей в середину компонент. Цю проблему вирішено за допомогою використання бібліотеки inversify-props, яка постачає глобальний контейнер для залежностей та дозволяє таким чином передавати залежності у vue-компоненти не зберігаючи самі компоненти в контейнері.

Щоб побачити важливість використання інверсії залежностей, розглянемо проблему, що виникає при розробці без використання інверсії залежностей та розберемо як саме реалізується даний шаблон. Розглянемо логічно пов’язану сукупність функціональних елементів як модуль. В контексті проєкту модулями будуть виступати клас HttpProvider, EquipmentHttpProvider та модуль vuex PrinterListStore тощо. Кожен з них має своє місце в ієрархії, так, наприклад, PrinterListStore залежить від EquipmentHttpProvider, оскільки користується екземпляром цього класу для виконання викликів на віддалений сервер. Своєю чергою EquipmentHttpProvider використовує сконфігурований HttpProvider для виконання викликів на віддалений сервіс на нижчому рівні, а на себе бере задачу опису інтерфейсу, додаткової обробки даних, що приходять та відправляються на сервер. Кожен з цих модулів використовує в процесі своєї

роботи інші модулі, таким чином формується деревовидна структура залежностей (Рис 4.4).

Використання такої структури породжує дві проблеми — між модулями існує великий ступінь зачеплення, що своєю чергою підвищує ризик виникнення помилок при зміні в якомусь з модулів на рівень вище.

Друга проблема виникає при тестуванні. Під час unit тестування тестується лише вихідний код конкретного класу чи функції, а всі її залежності повинні бути замінені заглушками, оскільки не беруть участь у тестуванні. Але в такому випадку стає питання того, як при тестуванні підміняти залежності. У випадку, якщо модуль самостійно ініціалізує залежності, то підмінити їх заглушками не можливо, оскільки код автоматизованих тестів не буде мати доступу на до них зовні.

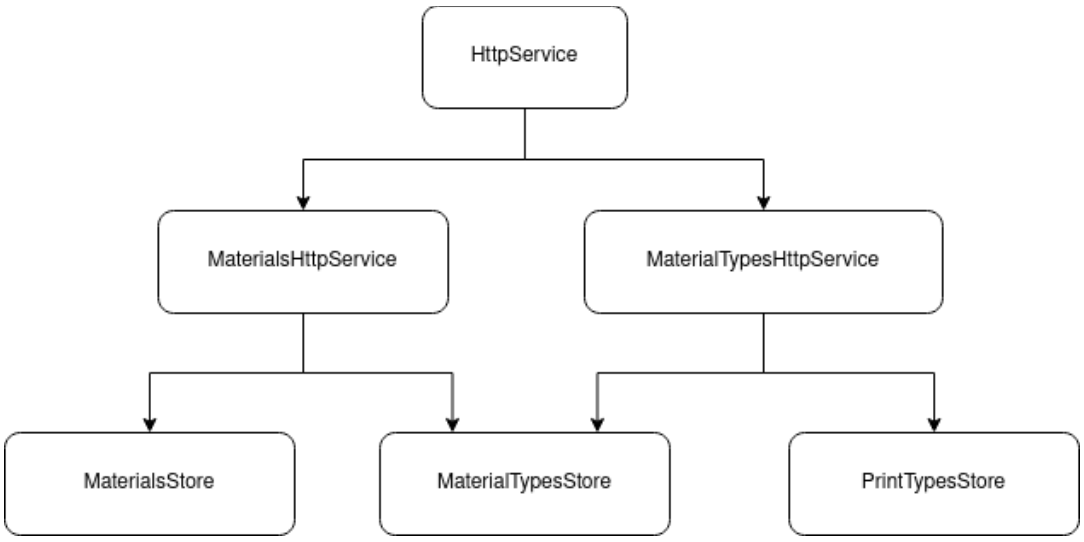

Рис 4.4 — Ієрархічна залежність між модулями. Стрілки вказують на напрямок залежностей між модулями.

Інверсія залежностей пропонує заміну залежності від реалізації залежністю від інтерфейсу. Це означає, кожен модуль в системі тепер розділяється на реалізацію та інтерфейс (рисунок 4.5).

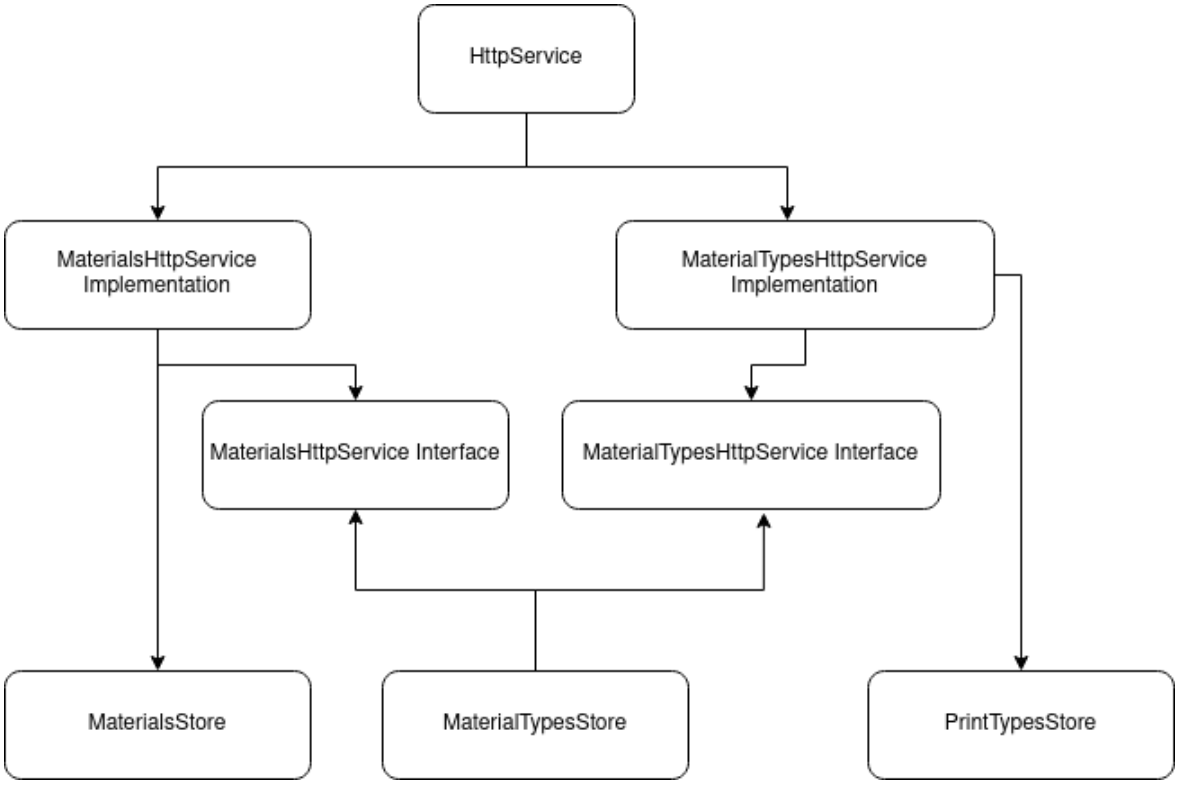

Рисунок 4.5 — Ієрархія модулів з використанням інверсії залежностей.

Але для того, щоб інверсія можливостей стала можливою виникає потреба в коді, який буде постачати потрібні залежності в середину модуля. На цей час існує багато різних бібліотек, що реалізують дану функціональність для різних мов. В проєкті використано дві різних системи для клієнтів та мікросервісів, оскільки фреймворк клієнта не надає готового рішення, тому для цієї задачі було обрано InversifyJs. Для мікросервісів фреймворк постачає власну систему інверсії залежностей, тому використовується вона.

Сама суть роботи контейнера залежностей (іос контейнер) полягає в тому, що для нього створюється конфігурація в якій до інтерфейсу прив'язується конкретна реалізація у вигляді класу чи фабрики, що створює екземпляр цього класу. Контейнер надає інтерфейс, через який зовнішній код може отримати екземпляр класу, що реалізує певний інтерфейс, в процесі ініціалізації цього класу іос-контейнер візьме всі потрібні залежності, якщо потрібно ініціалізує їх та передасть їм їх вкладені залежності, після чого поверне готовий екземпляр

класу.

Зазвичай іос-контейнери також надають додаткові функції, що роблять розробку зручнішою. Такою функцією виступає режим зберігання залежності. Наприклад, при потребі можна через іос-контейнер зробити клас Singleton і його екземпляр не буде створюватись кожен раз, коли його будуть діставати з контейнера у якості залежності, а буде ініціалізуватись лише один раз.

4.12 Висновки розділу

В даному розділі були описані основні технології, підходи та шаблони програмування, що використовувались при розробці додатку. Також були частково проаналізовані їх аналоги та обґрунтовано вибір сами цих технологій та підходів.

Вибір однієї мови для розробки як серверних додатків, так і для графічних клієнтів спрощує їх розробку та підтримку.

5 РОЗРОБКА СИСТЕМИ

При розробці вебсервісів, двома основними частинами додатку виступає сервер та клієнт. Дві ці частини беруть на себе різні ролі: сервер відповідає за зберігання даних, керує основною бізнес-логікою та являє собою закриту для зовнішніх користувачів систему, що спілкується з користувачем через чітко визначений інтерфейс та користується для цього різними протоколами, найбільш популярними з яких є HTTP, SOAP, Websocket. Клієнтська частина відповідає за надання користувачам інтерфейсу для роботи з сервером. Клієнт зазвичай являє собою графічний інтерфейс, але може мати будь-який вигляд, наприклад бути консольним додатком.

5.1 Мікросервісний підхід

Серверна частина додатку, яка відповідає за основну бізнес-логіку роботи сервісу використовує підхід з розбиттям додатку на мікросервіси, тобто окремі додатки, які розташовуються на окремих машинах. В наш час все більше додатків починають використовувати даний підхід оскільки він дає деякі переваги, яких позбавлені додатки-моноліти, а саме: Розробку різних частин додатку можна доручати різним командам, які не тісно між собою пов’язані. Розмір команд також можна масштабувати в залежності від потреб. так наприклад один сервіс може розробляти 10-20 людей, а може і одна, а іноді на команду з одного чи більше програмістів можна відвести одразу декілька сервісів.

Мікросервіси не пов'язані між собою напряму, тому можуть розташовуватись на різних платформах, наприклад, один сервіс буде розгорнуто на Amazon Web Services і він буде працювати на платформі NodeJs, а інший сервіс буде розгорнуто на Azure та буде працювати на платформи .NET Core. Такий підхід дозволяє зручно розділяти роботу над великим сервісом між

командами, які використовують різний підхід, а також обирати технології, які ідеально підходять для задачі, що вирішує конкретний мікросерівіс.

Оскільки мікросервіси розподілені, то це дозволяє при виході одного з сервісів із ладу не припиняти роботу всього додатку. Таким чином, якщо мікросервіс для коментарів вийде з ладу, то все інше продовжить працювати. Також зручно масштабувати різні частини додатку, розгортаючи більше екземплярів конкретного мікросервісу: якщо на сервіс для завантаження моделей йде більше велике навантаження, то можна розгорнути додаткові екземпляри цього додатку, щоб справлятись з потрібним навантаженням. Зараз додатки для публікації та розгортання свого продукту використовують сервіси для CI/CD(Continious intergration/Continious delivery), ці сервіси займаються автоматизацією тестування та розгортування, публікації нових версій ПЗ. При використанні мікросервісного підходу при зміні однієї частини додатку (мікросервісу) буде заново розгортатись лише один мікросервіс, що зменшує шанс того, що після розгортування перестане функціонувати весь додаток.

При використанні мікросервісного підходу потрібно вирішити ряд задач, пов’язаних зі спілкуванням сервісів між собою, розподілом відповідальності між ними, способами аутентифікації та авторизації користувачів при роботі з ними.

5.2 Сервіс автентифікації

Автентифікація — це процес визначення системою особи, що з нею працює на основі представленого нею ідентифікатора.

Основним способом автентифікації користувачів є перевірка системою ідентифікатора користувача, наприклад логіну чи адреси електронної пошти, та пароля, що був створений самим користувачем, чи виданий йому при реєстрації його облікового запису.

Автентифікація поділяється на слабку, яку ще називають однофакторною та сильну, або так звану багатофакторну автентифікацію.

Слабка автентифікація базується на перевірці пароля користувача. Такий метод не є повністю надійним, оскільки пароль можна перехопити при передачі його на сервер чи підібрати. Також впливає людський чинник, оскільки людина може сама передати пароль, чи зберігати його там, куди може отримати доступ інша людина. Окрім викрадання паролю, при його підборі людський чинник також має вплив, оскільки більш надійні та складні для підбору паролі важче запам’ятати, том люди роблять паролі або легкими для підбору та запам'ятовування, чи записують складні паролі там, де до них можуть отримати доступ треті особи.

Сильна автентифікація базується на складніших способах перевірки особи, які застосовуються додатково. Основними додатковими способами перевірки особи є:

- перевірка знань, якими володіє суб’єкт (додаткові запитання тощо);

- перевірка власності, що належить користувачеві (мобільний телефон, смарткарта тощо);

- властивість, якою володіє суб’єкт (біометричні характеристики, такі як відбиток пальця, сітківки ока, обличчя).

При двофакторній автентифікації у вебсистемі найбільш розповсюдженими є:

- перевірка за номером телефона, коли власнику телефона надсилають тимчасовий пароль, який потрібно ввести для підтвердження особи, чи телефонний дзвінок.

- перевірка за відбитком пальця чи іншими біометричними характеристиками: оскільки з вебсистемами можна взаємодіяти через смартфон, то використовуючи мобільний додаток з доступом до сканера відбитків чи обличчя можна підтвердити особу користувача.

- тимчасовий пароль у сторонньому сервісі: існують сервіси для подвійної

автентифікації, такі як наприклад рішення від компанії Google та інших.

Виходячи з цього, можна побачити, що для реалізації автентифікації. потрібно зберігати інформацію про користувачів та їх паролі, також при спілкуванні з сервісом користувач повинен проходити автентифікацію лише один раз, тому потрібен механізм для підтримання сесії. Тому потрібно виділити два мікросервіси: перший мікросервіс буде займатись зберіганням даних, а другий - видачею та перевіркою токенів.

В результаті реалізація автентифікації виглядатиме наступним чином:

- Користувач звертається до сервісу автентифікації та передає йому пароль та логін (ідентифікатор користувача).

- Сервіс автентифікації звертається до сервісу користувачів та отримує від нього дані про користувача, включаючи його данні про його пароль

- Після чого, якщо користувач з таким ідентифікатором існує, то сервіс авторизації звіряє паролі і, якщо вони збігаються, формує токен доступу та повертає його користувачеві.

Токен доступу — це тимчасовий ключ, який буде передаватись користувачем при подальшому спілкуванні з сервером для підтвердження своєї особи.

Токен буде сформовано за стандартом JSON Web Token (JWT).

JWT - це стандарт токена доступу, що базується на форматі JSON і використовується для верифікації звернень до сервера.

Токен, сформований за стандартом JWT, складається з трьох частин:

- Заголовок

- Вміст

- Підпис

Заголовок — це елемент, що описує тип токену та алгоритм шифрування, що використовувався для його створення.

В даному випадку тип токену - це application/jwt. Алгоритми шифрування в цьому стандарті зазвичай це HMAC(HS256) або RSA(RS256), що являють

собою геш-функції з сімейства SHA-2.

Вміст — це частина, що зберігає в собі інформацію про користувача, в даному проєкті вона має наступну структуру (рисунок 4.6).

В цій структурі поле exp - це дата закінчення дії токену, записана у форматі Unix.

Підпис - це частина, яка використовується для перевірки коректності токену. Вона формується шляхом кодування заголовка та підпису в Base64, після чого отримані частини конкатенуються через крапку, та шифруються обраним алгоритмом шифрування з використанням секретного ключа.

В результаті три отримані частини конкатенуються через крапку.

Такий токен зручно використовувати для автентифікації, передаючи його в заголовках Http запитів в полях Authorization чи Coockie. В даному проєкті обрано передачу токена в заголовку Authorization.

Рисунок 4.6 — Структура вмісту токена у форматі JSON.

У такого формату токену є один недолік: токен зберігає в собі термін завершення, тому немає уніфікованого рішення для випадку, коли користувач хоче деактивувати токен. У цьому випадку найпростішим рішенням є зберігання чорного списку токенів, які було деактивовано, і додатково перевіряти токен на предмет того, що він знаходиться в цьому списку при

перевірці токену.

5.3 Автентифікація та рівні доступу в мікросервісах

Оскільки додаток складається з окремих додатків, що розміщені окремо один від одного, то користувач для роботи з різними частинами додатку звертається до різних мікросервісів і тому він повинен мати можливість, пройшовши автентифікацію лише один раз, використовувати отриманий токен для підтвердження особи у всіх сервісах. Таким чином сервіс автентифікації завжди буде взаємодіяти з іншими сервісами. Для цього кожен сервіс використовує клієнт, через який звертається до сервісу автентифікації для перевірки актуальності токена та отримання даних про користувача від сервісу. Оскільки токен зберігає собі данні про користувача, що необхідні для роботи в середині мікросервісів, то додаткових звернень до сервісу користувачів не потрібно., оскільки сервіс автентифікації сам розшифровує та повертає данні про користувача.

Також важливим моментом при взаємодії користувача та мікросервісів є розподіл прав або так звана авторизація.

Авторизація — це процедура з’ясування чи має користувач права на виконання тої чи іншої операції в системі. Кожен мікросервіс сам керує цим процесом, базуючись на даних про користувача, що він отримує під час перевірки токена.

Для розділення прав між користувачами було вирішено використати підхід з розділенням користувачів по ролях. Всього в системі є три види користувачів: користувач, адміністратор та головний адміністратор (root). Роль користувача відноситься до самих користувачів сервісу, які завантажують моделі, переглядають каталог моделей, замовляють друк тощо.

Роль адміністратора полягає в роботі з модерацією моделей, налаштуванням інформації про актуальне обладнання, доступні матеріали та їх

види, види друку, а та отримують список моделей для друку та актуалізують статуси виконання замовлень.

Головний адміністратор може виконувати всі ті ж операції, що і адміністратор, але також має деякі особливі можливості. В першу чергу — це створення та блокування облікових записів адміністраторів, оскільки система дозволяє самовільно зареєструватись лише звичайним користувачам, а процес створення нових ролей адміністраторів відводиться ролі головного адміністратора.

5.4 Зберігання файлів

Оскільки важливою частиною роботи з сервісом є завантаження моделей, розроблених в CAD-системах, а також зображень, було вирішено виділити цю функцію в окремий мікросервіс.

Сервіс має місце для зберігання файлів та базу даних, яка зберігає в собі данні про завантажені моделі, для того, щоб можливо було розділяти моделі на публічні та приватні.

Сервіс для зберігання файлів може мати два види взаємодії: з авторизацією та без. Для завантаження моделей можна використати спосіб завантаження з авторизацією, в такому випадку сервіс, при отриманні файлу, перевірить токен і поверне файл тільки у випадку, якщо файл був завантажений користувачем, що його запитує, або, якщо цей файл запитує адміністратор.

Для завантаження зображень матеріалів, принтерів, моделей тощо можна використовувати спосіб завантаження без авторизації: в такому випадку сервіс зберігає данні про автора файлу, але повертає його будь-кому, хто буде намагатись його завантажити.

При завантаженні будь-якого файлу його оригінальне ім’я замінюється на згенероване випадковим чином слово, що сформовано с 64-х чисел від 0 до F у шістнадцятковому форматі. Це додатковий спосіб захисту файлів, для того, щоб

з сервера не звантажували файли методом перебору числових ідентифікаторів.

5.5 Мікросервіс для обладнання

Мікросервіс для обладнання бере на себе зберігання даних про обладнання, доступні матеріали, а також планування друку та формування приблизних оцінок вартості друку та термінів виконання замовлення. Сервіс має як інтерфейс для звичайних користувачів так і інтерфейс для адміністратора.

При аналізі параметрів, які потрібні для формування замовлення було визначено, що основними параметрами для друку є матеріал, тип принтера, на якому буде друкуватись модель, відсоток заповнення моделі всередині та лінійні розміри моделі. Для того, щоб на основі цих даних можна було сформувати приблизну вартість та строки друку було проаналізовано наявні моделі принтерів та особливості друку з їх використання та було з’ясовано наступне:

— існує більше ніж сім видів 3D друку;

— окрім технологічних особливостей процесу друку на різних принтерах результат їх роботи розрізняється в першу чергу матеріалами виробу та розмірами моделей на які розраховані принтери;

— більшість моделей одного і того ж типу зазвичай розрізняються розмірами, а якість друку на принтері в основному залежить від швидкості друку;

— витрати матеріалу не залежать від моделі принтера.

На основі цих фактів можна виділити таблиці в базі даних, з якими має працювати мікросервіс:

— типи матеріалів,

— конкретні матеріали,

— типи друку,

— моделі наявних принтерів,

— наявні принтери

— робочі дні принтера

Оскільки інформація про обладнання в першу чергу розрахована на те, щоб надати користувачам опції для формування замовлення, то дані про типи матеріалів носять інформативний характер, який потрібен користувачеві для того, щоб зрозуміти який матеріал обрати, тому таблиця матеріалів в базі даних має мати колонки назви, опису та посилання на зображення матеріалу та прикладів виробів з нього, але також зберігаються колонка з вагою матеріалу на кубічний сантиметр, що буде враховано при розрахуванні вартості друку.

Таблиця матеріалів зберігає інформацію про тип матеріалу та його колір, що записується парою значень — назва кольору та його в форматі RGB для показу в списку опцій під час вибору матеріалу на стороні клієнта. Всі інші потрібні дані про матеріал можна дізнатись з його типу. Оскільки існує багато видів друку на 3D принтері, які можуть додаватись з часом, а також відрізняються між собою типами матеріалів, що використовуються при друці на принтерах цього типу, типи принтерів були винесені як окрема таблиця в базі даних сервісу.

Таблиця моделей 3D принтерів зберігає в собі вичерпні дані про модель та містить наступні колонки:

— назва моделі,

— опис,

— зображення,

— мінімальна можлива довжина моделі (см)

— максимальна можлива довжина моделі (см)

— мінімальна можлива висота моделі (см)

— максимальна можлива висота моделі (см)

— мінімальна можлива ширина моделі (см)

— максимальна можлива ширина моделі (см)

— низька швидкість друку (кг/сек)

— середня швидкість друку (кг/сек)

— висока швидкість друку (кг/сек)

— вартість друку за годину

— тип друку

Такий набір параметрів зумовлений наступними факторами:

— колонки назви, опису та зображення — це інформаційні поля, що потрібні для зручності налаштування системи адміністраторами;

— колонка лінійних розмірів потрібні для того, щоб при розподіленні моделей на друк принтерами можна було обрати лише відповідні моделі принтерів, а також мати можливість підібрати найменший принтер для кожної моделі, таким чином зменшивши вартість друку, адже менші принтери зазвичай дешевші і це відповідно впливає на вартість друку. Також мінімальні розміри вказані тому, що на деяких моделях принтерів дрібні деталі мають низьку якість і використання їх таким чином є нераціональним;

— швидкість друку може досить сильно варіюватись в залежності від програмного забезпечення, що використовується для друку моделі, тому було вирішено використовувати для вибору швидкості друку опції “повільно”, “середня швидкість” та “висока швидкість” для того, щоб ці дані адміністратор системи міг вказати в налаштуваннях моделі принтера. Швидкість вказується як середнє значення в кілограмах за секунду;

— вартість друку описується в умовних грошових одиницях за секунду принтера. Цей показник враховується при розрахуванні вартості друку на конкретній моделі принтера.

— також принтер має посилання на тип друку, що потрібно для того, щоб зрозуміти який тип друку використати в залежності від обраного типу матеріалу для друку.

Структуру таблиць бази даних мікросервісу наведено в ІА62.190БАК.005 Д3.

Мікросервіс обладнання має в собі достатньо даних для того, щоб визначити приблизну вартість друку конкретної моделі, тому цю функцію було відведено йому.

Окрім розрахунку вартості та надання інформації про матеріали, даний мікросервіс містить функціональність для планування процесу друку. Він має внутрішній інтерфейс для мікросервісів при формуванні замовлення на основі переданих даних формує план друку. Оскільки виробництво не розраховано на цілодобову роботу для формування плану дурку виділено окрему таблицю, яка об’єднує моделі для друку, принтер на якому вони мають бути видруковані та день тижня. При формуванні плану мікросервіс враховує тривалість робочого дня і формує план таким чином, щоб час використання кожного принтера в день не перевищував стандартного робочого дня.

Створення замовлення потребує декількох етапів взаємодії між мікросервісами, оскільки це потрібно для формування зрозумілих помилок з відповідними для кожної ситуації кодами помилки. Діаграму послідовності взаємодії мікросервісів при формуванні замовлення наведено в ІА62.190БАК.005 Д4.

5.6 Робота з моделями

Робота моделями виділена в окремий сервіс, який має свою окрему базу даних. Призначенням сервісу в першу чергу є створення інформації про моделі в базі даних. Він представляє інтерфейс для створення моделей, їх публікації користувачами, а також надає адміністраторам інтерфейс для модерації моделей. Також сервіс повертає інформацію про моделі для різних контекстів: власні моделі для користувача, каталог опублікованих моделей та моделі, що потребують модерації.

Процес модерації є дуже важливим, оскільки адміністратор не тільки перевіряє модель на коректність, а й збирає данні про її об’єм та пропорції для

того, щоб у майбутньому можна було формувати ціну друку виробу в залежності, від вказаних користувачем опцій, базуючись на заданій при модерації інформації.

Як формат для завантаження моделей було обрано формат STL. STL — це формат, який дуже часто використовується у якості формату для збереження 3D моделей, розроблених в CAD-системах і призначених для 3D друку та інших способів перетворення їх в тривимірні вироби. Для зберігання інформації про модель використовуються трикутні полігони. Для опису полігонів використовують три точки, що формують трикутник та нормаль. Таким чином для кожного трикутника потрібно 12 чисел з рухомою крапкою довжиною 32 байти кожен.

Файли даного типу можуть бути текстовими з використанням ASCII кодів для опису значень та двійковими. Оскільки при використанні ASCII символів файл може сильно збільшуватись в об’ємі, то було вирішено використовувати і двійковий варіант для збереження даних, що допомагає зменшити розмір отриманого файлу.

Формат не надає універсального рішення для зберігання кольору і різні CAD-системи по-різному реалізують дану функцію:

— системи SolidView та VisCam використовують додаткові два байти для опису кольору полігону в палітрі RGB.

— система Materialize Magics використовує заголовок файлу для збереження додаткової інформації про загальний колір моделі та її матеріал.

Відсутність в форматі кольору за замовчуванням є корисною рисою для даного проєкту, оскільки моделі з текстурами, такі, як, наприклад, .obj мож можуть бути складнішими для оцінки їх модератором, а також зображення зроблені для демонстрації моделі користувачу з текстурами можуть заважати користувачеві зрозуміти приблизний результат, який буде отримано при замовленні друку цієї моделі.

Основним недоліком формату є досить великий розмір вихідного файлу

для складних моделей, що містять багато полігонів.

Для спрощення пошуку користувачами 3D моделей різного типу було введено додаткову таблицю категорій, що містить в собі назву категорії, її опис та зображення. В головному каталозі сайту буде як перегляд всіх моделей, та і розбиття та їх фільтрація по категоріях. Створення, редагування та видалення категорій буде здійснюватись головним адміністратором системи.

Виходячи з отриманої структури мікросервісу обладнання можна побачити, що спроєктована структура підходить не тільки для адитивних технологій, представлених 3D друком, а й для замовлення виробів, що виробляються на ЧПК станках, оскільки:

— система дозволяє створювати різні типа матеріалів і в неї можна додати різні види деревини, металів тощо;

— Параметри, відібрані для опису моделей 3D принтерів також підходять і для опису ЧПК станків, оскільки вони також характеризуються лінійними розмірами, матеріалами, які можливо використати для обробки на станках, а також швидкість та ціна за час роботи;

Але як можна помітити при замовленні друку на 3D принтері важливим фактором, що впливає на якість отриманого виробу та його вартість є швидкість друку: чим нижча швидкість друку - тим вища якість, але вища і ціна, адже швидкість може змінюватись в декілька разів. Також для розрахунку витрат матеріалу виробів, виготовлених на ЧПК станках не підходить спосіб розрахунку через розрахування об’єму отриманого виробу: ЧПК станок, на відміну від адитивних технології, формує деталь шляхом знімання шарів матеріалу з бруска.

Враховуючи вище вказані факти було вирішено розширити функціональність категорій в сервісі моделей. Для чого в таблицю категорії були додані дві додаткових інформаційних колонки.

Перша з них — це стратегія розрахунку витрат матеріалу, яка має дві опції: “shape” та “box”, при використанні першої опції система буде розраховувати

кількість матеріалу, виходячи з об’єму отриманої деталі, а при використанні другої системи буде брати за основу об’єм паралелепіпеда, отримані шляхом множення найбільших ширини, висоти та довжини моделі. Також система при розрахунку кількості витрачених матеріалів повинна враховувати заповнення моделі, але у випадку з виробами на ЧПК станках коефіцієнт заповнення завжди буде дорівнювати 100%.

Для того, щоб користувачу не потрібно було завжди вказувати коефіцієнт заповнення та швидкість, які завжди будуть однаковими для моделей, призначених для друку на ЧПК станках, то в таблицю категорій також додано колонка, що містить в собі об’єкт, що описує поля за замовчуванням для моделі з даної категорії. Значення за замовчуванням мають більший пріоритет над тими, що міг би вказати користувач і таким чином дозволять при потребі розширити асортимент послуг сервісу, включивши в нього виготовлення деталей на ЧПК станках з деревини, металів тощо.

5.7 Мікросервіс замовлень

Керування процесом формування замовлення виступає мікросервіс для замовлень. Його основна задача полягає в надані інтерфейсу для створення користувачем замовлення, валідації замовлення на предмет коректності даних по замовленню та взаємодії з мікросервісом обладнання для того, щоб передати йому інформацію, що потрібна для реалізації замовлень.
При формуванні замовлення користувач повинен передати масив елементів, кожен з яких містить такі данні як:

— ідентифікатор моделі, що друкується,
— висота моделі,
— ширина моделі,
— довжина моделі,
— матеріал для моделі

— коефіцієнт заповнення (від 20% до 100%),

— швидкість друку

Ці данні передаються також потребуються сервісу для обладнання, тому даний мікросервіс має інтерфейс для отримання даних замовлення іншим мікросервісом обладнання.

5.8 Мікросервіс користувачів

Мікросервіс користувачів бере на себе задачу зберігання даних про користувачів: як адміністраторів так і звичайних користувачів. Також сервіс зберігає інформацію про адреси доставки, створені користувачем для того, щоб отримувати на них замовлення.

5.9 Висновки розділу

В даному розділі було описано основні деталі реалізації системи на прикладі її мікросервісів та графічних клієнтів. В результаті було спроєктовано шість мікросервісів, що являють собою окремі додатки, а також два графічних клієнти, що потрібні для роботи операторів та користувачів з системою

6 ТЕСТУВАННЯ СИСТЕМИ

Зазвичай тестування програмного продукту - це невід'ємна складова процесу його розробки. Тестування програмного продукту можна розділити два види: ручне тестування та автоматизовані тести. Дані два способи доповнюють один одного та допомагають виключити основний спектр помилок, які потенційно допущені при розробці програмного забезпечення. Нижче наведено огляд цих способів, їх аспектів та приклади їх застосування до розробленого програмного продукту.

6.1 Ручне тестування

Ручне тестування - являє собою спосіб тестування, при якому людина, за певним сценарієм вручну перевіряє правильність роботи програмного забезпечення, шляхом тестування їх інтерфейсів, що можуть бути представленні різними способами взаємодії з програмним забезпеченням. Основною перевагою цього способу є те, що людина під час тестування може зіштовхнутися з будь-якими проблемами та в першу чергу важливо те, що людина можне помітити ті аспекти та недоліки, які можна не врахувати при розробці автоматизованих тестів. Таким чином такий підхід в першу чергу підходить для тестування складних сценаріїв, особливо для графічних інтерфейсів, де окрім досягнення кінцевого результату важливо перевірити коректність роботи графічного інтерфейсу в процесі роботи, адже багато інструментів для автоматизованого тестування зазвичай можуть не виявити такі проблеми як мерехтіння компонентів графічного інтерфейсу, неправильні пропорції чи розміщення елементів графічного інтерфейсу, оскільки більшість автоматизованих інструментів в першу чергу працюють з кодовою базою та орієнтуються на те, що елемент присутній, його можливо побачити та з ним взаємодіяти, але при цьому ігнорує багато аспектів взаємодії з ним, що

приведено вище, або потребує занадто великих витрат на те, щоб написати тести, які можуть це виявити та підтримувати їх.

Ручне тестування компонентів додатку проводилось протягом усього часу розробки проєкту. Об’єктів для тестування є два: графічний інтерфейс, що доступний через браузер та програмний інтерфейс додатку, доступний через протокол HTTP.

6.1.1 Ручне тестування програмного інтерфейсу

Під час розробки мікросервісів може бути дуже не зручними тестування їх через графічний інтерфейс, оскільки при такому підході потрібно розробляти графічний інтерфейс в процесі розробки серверного програмного забезпечення, а це може значно сповільнити роботу.

Для тестування програмного інтерфейсу серверного додатка, що працює через протокол HTTP використовувався REST клієнт Insomnia. Дане програмне забезпечення дозволяє створити каталог з заготовлених викликів до програмного інтерфейсу, дозволяє відправляти дані в багатьох форматах, керувати змінними оточення та автоматизувати процес обробки відповідей від сервера.

Якщо розглядати інтерфейс додатку, то можна виділити три основні його частини: каталог запитів (рисунок 6.1), вікно редагування запиту (рисунок 6.2), вікно перегляду відповідей та меню для управління оточенням.

Каталог запитів розбито на шість основних директорій, кожна з яких відповідає за окремий мікросервіс, своєю чергою каталог мікросервісу містить підкаталоги, що групують запити до кожного окремого контролера. Таке розбиття зрозуміле та очевидне, що спрощує ручне тестування через даний клієнт. Також присутній вибір оточень, що корисно у випадках, якщо програмне забезпечення тестується в різних оточеннях, наприклад локально та на віддалених серверах. Змінні оточення записуються у форматі JSON.

На рисунку 6.2 показано, що кожен запит містить в собі багато налаштувань, основними з яких є метод запиту, тіло запиту, якщо воно присутнє, у багатьох популярних форматах, таких як JSON, XML, YAML, вебформа тощо. Також налаштовуються заголовки в тому числі заголовки авторизації.

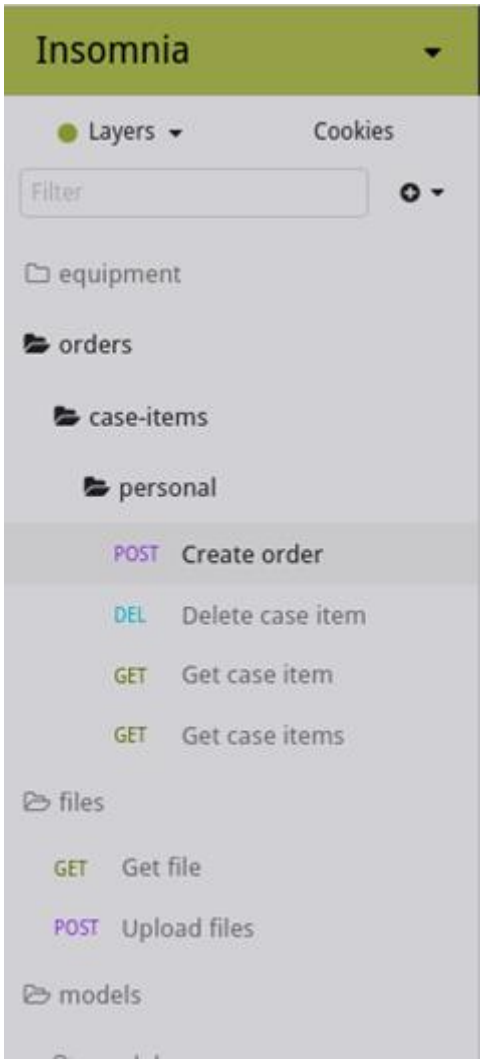

Рисунок 6.1 — Каталог запитів

Оскільки для авторизації у всіх мікросервісах використовується єдиний токен, то дуже довго та незручно кожен раз вручну задавати токен, що згенеровано мікросервісом авторизації. Тому даний додаток надає механізм для автоматичного зберігання токену, отриманого у відповіді сервера при виконанні запиту авторизації. Таким чином для початку роботи з усіма

запитами до мікросервісів достатньо виконати лише один запит на авторизацію.

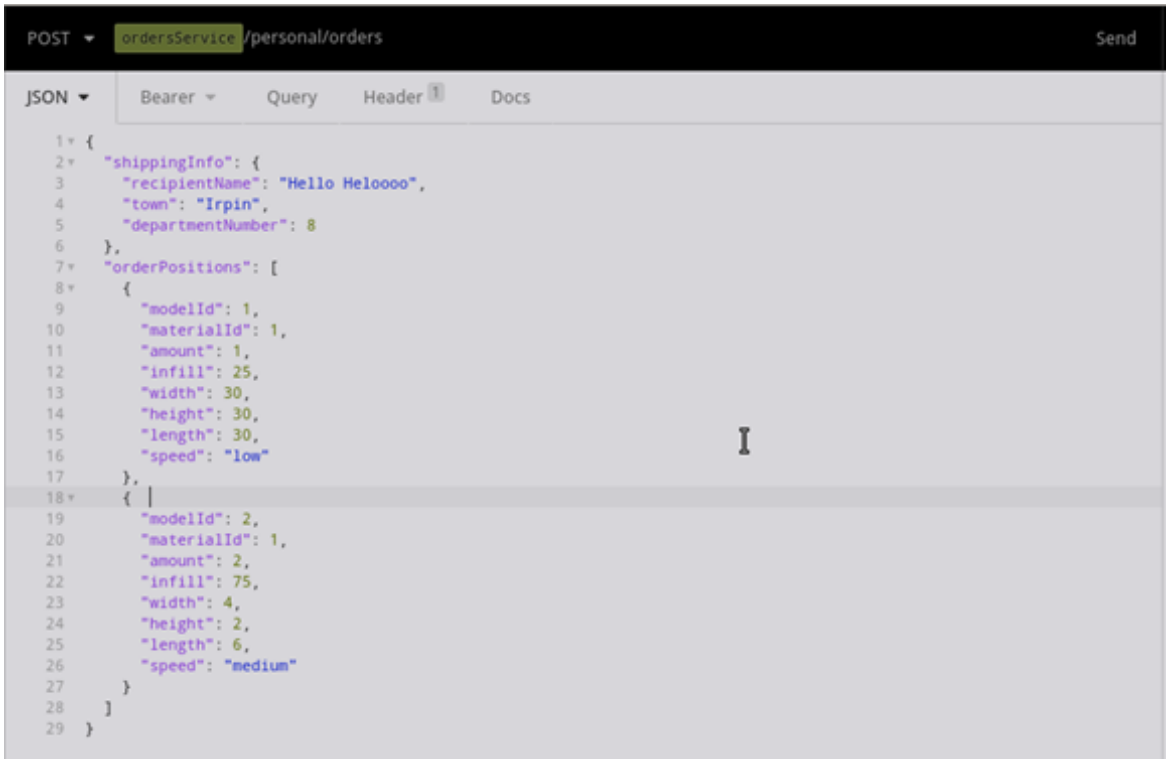

Рисунок 6.2 — Вікно налаштування запиту

6.1.2 Ручне тестування графічного інтерфейсу

Ручне тестування графічного інтерфейсу зазвичай проводиться як самим розробником в процесі його розробки, так і тестувальником, що повинен перевірити його на предмет відповідності функціональним вимогам, вимогам до коректності виконаної роботи в технічному плані.

Ручне тестування проводилось протягом усього часу розробки графічного інтерфейсу та після його завершення. Оскільки даний інтерфейс відкривається через браузер, то при тестуванні важливими були наступні вимоги:

— інтерфейс повинен працювати коректно у вікні браузера розміром до 992 пікселів;

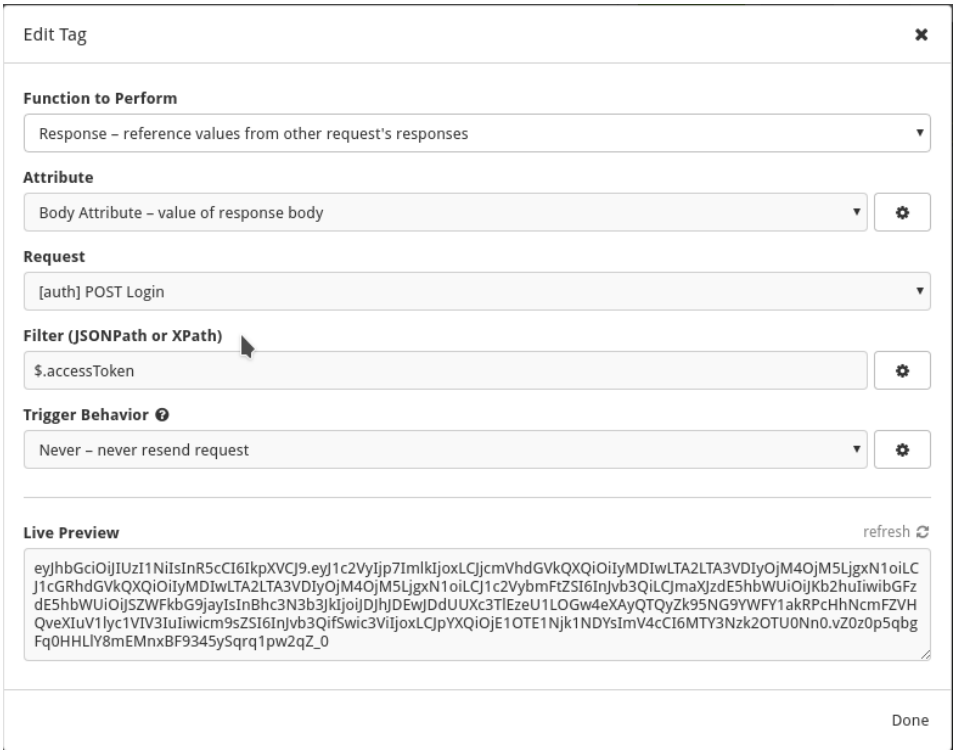

Рисунок 6.3 — Налаштування для автоматичного зберігання токена

— інтерфейс повинен коректно працювати в найбільш популярних браузерах таких як Mozilla Firefox, Google Chrome, Chromium та Opera.

Найбільш популярні браузери пропонують інструменти для тестування та налагодження графічних клієнтів. Приклад тестування буде наведено для браузера Mozilla Firefox.

Щоб відкрити інструменти розробника потрібно перейти на сторінку та натиснути клавішу F12, або натиснути праву клавішу миші та обрати пункт «Inspect Element», після чого відкриється вікно з інструментами розробника. На приведеному рисунку для тестування було використовувати інструменти на вкладках Network, Storage та Perfomance.

Вкладка Network дозволяє переглядати всі виклики на віддалені ресурси що виконуються сторінкою. При тестуванні перевірялась коректність XHR (XMLHttpRequest), що відправляє сторінка, а саме правильність параметрів, відповіді, адреси звернень та відповіді, все це можна побачити через дані інструменти.

Вкладка Storage відповідає за перегляд даних, що зберігаються в Local Storage, Session Storage, Cookies. Перевірка цих даних була потрібна при розробці сервісу LocalStorageService для графічного клієнта, що виступає посередником між викликами до інтерфейсу браузера та кодом додатку.

Вкладка Perfomance дозволяє відстежувати процес виконання програми, ресурси, що витрачає сторінка на процес візуалізації контенту та проведення обрахувань.

6.2 Автоматизоване тестування

Автоматизовані тести зазвичай пишуться паралельно з розробкою коду додатку. Існує три основних види автоматизованих тестів: модульні тести, інтеграційні тести та end-to-end тести.

Для розробки модульних та інтеграційних тестів в проєкті використовується бібліотека Jest, яка надає інструменти для запуску тестів на платформі NodeJs (як і коду клієнтів, та і коду мікросервісів), перевірки тверджень, також бібліотека надає програмні компоненти, що дозволяються робити заглушки методів та функцій, також за замовчуванням бібліотека при проходженні тестів дозволяє імітувати інтерфейси браузера.

Додатково для тестування компонентів, що роблять виклики на віддалений вебсервер за протоколом HTTP використано бібліотеку SinonJs, яка дозволяє перехоплювати виклики на віддалений сервіс та заміняти їх власними заглушками.

6.2.1 Модульні тести

Модульні тести для даного продукту відповідають за тестування коду на найбільш низькому рівні, оскільки тестують виключно логіку роботи окремих компонентів без залежності від оточуючого коду.

Модульні тести повинні бути написані в найбільшій кількості, оскільки це найпростіший та досить надійний спосіб тестування коду, адже модульні тести прості за своєю будовою та не потребують багато часу на написання.

При написанні модульних тестів сам тест перевіряє публічний інтерфейс компонента програми, не прив’язуючись до його реалізації. Це дозволяє надалі переписувати код програмних компонентів з метою покращення його якості, швидкості виконання чи надійності, при цьому використовуючи модульні тести для перевірки того, що логіка роботи компонента залишилася незмінною.

Для того, щоб мати повний контроль над виконанням тесту потрібно мати повний контроль над всіма залежностями компонента програми, що тестується. У пункті 4.1, де розглядалась інверсія управління, було сказано, що цей підхід дозволяє реалізувати підміну залежностей програмних компонентів. Для цього під час тесту для нього створюється окремий контейнер з залежностями, що зберігає в собі об’єкти, що реалізують інтерфейси, які потребує компонент, що тестується, але замість реальної реалізації всі їх методи та поля заміняються заглушками, якими можна керувати в процесі виконання коду тестів.

6.2.2 Інтеграційні тести

Інтеграційні тести в даному додатку беруть на себе задачу тестування взаємодії між компонентами програми. Якщо у випадку з модульними тестами кожен компонент тестується ізольовано, то при написанні інтеграційних тестів тестується група компонентів, що пов’язана між собою.

В технічному плані розробка інтеграційних тестів дуже схожа на розробку модульних тестів, але на відміну від модульних тестів, в контейнер з залежностями передаються реалізації класів, а не заглушки. Але все ж таки при розробці інтеграційних тестів було використано заглушки, оскільки клієнтські тести виконуються на серверній платформі, то виклики до інтерфейсу

веббраузера та віддалених серверів все ще підміняються заглушками або симулюються бібліотекою Jest.

6.2.3 End-to-end тестування

End-to-end тестування відноситься до найскладніших для реалізації та найдорожчих у підтримці видів тестів, оскільки:

— цей вид тестів потребує розгортання додатку та його інфраструктури для проходження тестів;

— цей вид тестів перевіряє чітко визначенні сценарії і при будь-яких змінах в сценарії їх потрібно переписувати;

— оскільки тести мають складний сценарій та потребують для своєї роботи запуску всього додатка, то вони набагато повільніші, ніж інші види автоматизованих тестів.

З огляду на вище приведені фактори, а також на той факт, що багато зі сценаріїв простіше перевірити шляхом ручного тестування було прийнято рішення відмовитись від написання end-to-end тестів. Також варто зазначити, що багато наявних інструментів, що наявні зараз для цього вид тестування не достатньо ефективні. Так наприклад при тестуванні додатку на віддалених пристроях можна зіштовхнутись з проблемою того, що тести можуть не пройти з вини сервісу, який надає пристрої для тестування.

6.3 Висновки розділу

В даному розділі було описано використані під час розробки підходи до тестування програмних додатків проєкту. Описані деталі та особливості тестування графічних клієнтів, а також обґрунтовано відмову від end-to-end тестування при розробці.

7 ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМИ

В пунктах нижче описано впровадження, розгортування та використання системи.

7.1 Розгортування системи

Оскільки система є розподіленою, то її впровадження та розгортання на першому етапі має буди одночасним, оскільки всі мікросервіси та клієнти пов’язані між собою і для реалізації всього функціоналу необхідно, щоб всі її компоненти були запущені. Після того, як система буде запущена, подальша розробка та підтримка система може бути частковою лише для того функціоналу, який потрібно розширити чи виправити. Оптимальним шляхом є розгортування системи в хмарних сервісах, таких як Amason Web Services чи Microsoft Asure.

Кожен додаток має свій власний набір змінних, які зберігають в собі посилання на інші сервіси та данні про розміщення самого додатка. При локальній розробці для передачі змінних у мікросервіси використовується docker-compose, який для кожного окремого додатка встановлює значення, задані для його файлу зі змінними оточення, що називається ".env" та лежить в кореневій директорії кожного додатку. У випадку з графічними клієнтами змінні оточення постачаються Webpack, та зберігаються в додатку на етапі компіляції. Оскільки кожен хмарний сервіс може мати свої власні способи задання змінних оточення, як наприклад GitLab CI, то адміністратор, що буде розгортати систему повинен задати їх власноруч. Для того, щоб дізнатись, які змінні оточення потребує кожен окремий додаток, потрібно переглянути файл ".env.dist", що знаходиться в корені директорії кожного додатка.

При локальній розробці база даних запускається в окремому контейнері

за допомогою docker-compose. При розгортанні робочої системи розміщення бази даних може бути довільним: це може бути як хмарний сервіс, що постачає лише доступ до бази даних і власноруч її обслуговує, або це може бути власний комп’ютер компанії, якщо вона бажає зберігати данні в себе чи на серверах, що розміщенні в сервісному центрі. Для підключення до бази даних потрібно в змінній оточення додатку передати такі данні, як адреса бази даних, пароль та ім'я облікового запису користувача, ім’я бази даних. Важливо, що при локальній розробці ORM додатку на етапі запуску видаляє всі данні. Щоб прибрати таку поведінку в файлі ".orm.config.ts", що присутній у мікросервісах, потрібно встановити поля synchronize та dropSchema у false, а в поле migrationsRun встановити значення true. Також кожен мікросервіс, окрім мікросервісу файлів має спеціальний сервіс, що записує в базу даних заготовленні дані. Це потрібно для її тестування. При розгортанні додатку потрібно вилучити цей файл зі списку модулів додатку в файлі main.ts.

Важливо, що для розгортання мікросервісу для файлів перед запуском додатку за допомогою команди чи іншого способу, що надає сервіс потрібно створити директорію з назвою uploads в корені директорії з якої буде запущено сам додаток і де буде розміщений він сам.

Запуск кожного мікросервісу потребує наявності на комп’ютері або в контейнері, на якому він буде запущений, платформи NodeJs версії 12.0. Перед запуском потрібно синхронізувати всі залежності, для цього платформа, на якій буде запущено додаток повинна мати доступ в мережу інтернет. Щоб запустити встановлення залежностей потрібно виконати команду «npm install». Запуск самого додатку виконується командою «npm run serve:prod». Після запуску додаток намагається під'єднатися до бази даних, тому сервер бази даних для кожного додатку повинен бути запущений до того, як буде запущено сам додаток.

7.2 Графічні клієнти

Локальна розробка графічних клієнтів виконується з використанням локального сервера, що постачається Webpack, який компілює код додатку після кожної зміни в файлах. Сама процес компіляції відбувається всередині docker контейнера, на ньому ж запускається сервер для локальної розробки. Для розгортання клієнтських додатків потрібно встановити залежності, виконавши команду «npm install», після чого скомпілювати код додатку, виконавши команду «npm run build», після чого файли, які буде згенеровано в директорію dist можна завантажити на статичний сервер, при зверненні на корінь якого, сервер має повернути файл index.html.

аРобота по налаштуванню додатка для роботи після цього запуску відбувається через графічний клієнт, що називається backoffice. При ініціалізації в системі існує лише один користувач - root. Цей користувач має найбільше прав в системі, він же займається створення облікових записів для операторів. Після реєстрації облікового запису оператора, йому потрібно передати його ім'я облікового запису та пароль, під яким він буде працювати в системі. Додаток для адміністратора має наступні сторінки для управління системою:

— Items to print - це список запланованих до друку моделей -Orders to sending - це список замовлень, які вже видруковані та повинні бути відправленні.

— Models moderation - список моделей, які потребують модерації. Про кожну модель надана інформація про її опис, зображення там сам файл моделі, який оператор повинен завантажити та впевнитись в тому, що модель можливо видрукувати. Використовуючи програмне забезпечення, яке використовується для друку моделей, адміністратор повинен провести замір крайніх точок моделі та її об’єму та вказати ці дані, щоб модель змогла бути замовлена для друку.

— Print types - це розділ в якому можна додавати нові види друку та зв’язувати їх в з наявними типами матеріалів в системі. - Material types - це сторінка для редагування інформації відносно типів матеріалів, які можна обрати при замовленні друку

— Materials - сторінка, що зберігає дані про матеріали доступні в системі, вона пов’язана з типом матеріалів та зберігає інформацію про щільність матеріалу та його колір.

— Printer models - сторінка на якій зібрано список моделей принтерів, що доступні в системі до яких можна додавати нові моделі, сказавши їх основні характеристики

— Printers - відповідає за конкретні принтери, що наявні в системі. При оформленні замовлень між ними на друк будуть закріплюватись моделі для друку. Після налаштування цих параметрів система буде готова для роботи і надалі основна робота операторів буде пов’язана з модерацією нових моделей та обробкою моделей для друку та їх виправлянням.

7.3 Висновки розділу

В даному розділі було розглянуто основні нюанси, з якими потрібно бути ознайомленим, перед початком впровадження додатка в роботу. Самим впровадженням повинні займатись спеціаліст з розгортування та адміністрування вебдодатків, а також головний оператор, що буде працювати з системою, оскільки одразу після запуску додатку його робота повинна бути налаштована людиною. Для налаштування системи надані інструменти, що знаходяться в графічному клієнті адміністратора.

ВИСНОВКИ

У результаті виконання даного дипломного проєкту було розроблено систему управління замовленням друку 3D-виробів на базі мікросервісів, що надає функціонал та засоби призначенні для її налаштування та використання.

Під час аналізу наявних рішень у якості оптимального стека технологій для реалізації системи було обрано мову Typescript в поєднанні з фреймворком NestJs. Отриманий програмний продукт реалізовано у вигляді сучасного та зручного вебсервісу, оскільки для його розробки були використано одні з найпопулярніших та актуальних на цей час інструментів. Усього було розроблено вісім додатків, два з яких є графічними клієнтами, а шість — серверними додатками. Було проведене тестування отриманих продуктів. Також, оскільки система є вебдодатком, бізнес-логіка роботи якого не залежить від графічних клієнтів, то надалі можливо розробляти додаткові графічні клієнти для мобільних платформ, таких як Andriod та iOS.

При більш детальному розгляді предметної області та, зокрема, особливостей виробництва продукції на 3D принтері, були закладені основи для подальшого розширення системи для роботи не лише з 3D друком, оскільки архітектура розробленої системи достатньо гнучка та розроблена такими чином, щоб в процесі її можна було легко конфігурувати та розширювати її можливості в області замовлення різних видів друку та розширення опцій, доступних користувачеві під час формування замовлення.

Також в процесі підготовки до роботи та аналізу наявних рішень у сфері замовлення друку виробів на 3D принтері було досліджено стан цієї сфери та підтверджено актуальність розробки та необхідність подальшого розвитку отриманого програмного продукту, оскільки на цей час адитивні технології активно розвиваються а сервіси для замовлення послуг онлайн стають стандартом в індустрії надання послуг та продажу товарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Томас Коннолли - Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Томас Коннолли, Каролин Бегг. . Вільямс м. Москва. 2014.
2. Резиг Джон - JavaScript. Профессиональные приемы программирования. Питер. 2008.
3. Алан Купер - Алан Купер об интерфейсе. Основы проектирования взаимодействия. Символ-Плюс. 2009.
4. Раскин Джеф - Интерфейс: новые направления в проектировании компьютерных систем. Символ-Плюс. 2005.
5. Каскиаро Марио - Шаблоны проектирования Node.js. / Каскиаро Марио, Маммино Лучано. ДМК Пресс. 2012.
6. Ганс-Юрген Шениг - PostgreSQL 11. Мастерство разработки. ДМК Пресс. 2019.
7. Свистунов, А. Н. Построение распределенных программных систем на Java / А.Н. Свистунов. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011.
8. Офіційна онлайн документація NestJs для версії 7.1.4
9. Офіційна онлайн документація VueJs для версії 2.3.7
10. Борис Бейзер - Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. Питер. 2004.
11. Гленфорд Майерс - Искусство тестирования программ / Гленфорд Майерс, Том Баджетт, Кори Сандлер. Вильямс. 2019
12. Бек Кент - Экстремальное программирование. Разработка через тестирование. Питер. 2020.
13. Мартин Роберт - Чистая архитектура. Искусство разработки программного обеспечения. Питер. 2018.

14. Мартин Роберт К. - Чистый код. Создание, анализ и рефакторинг. Питер. 2019.

15. Мартин Фаулер - Шаблоны корпоративных приложений / Мартин Фаулер при участии Девида Райса, Метью Фоммела, Едварда Хаета, Роберта Ми, Ренди Стаффорда. Вильямс. 2014.